4ª Edición



INTRODUCCIÓN A LA INFORMÁTICA

ALBERTO PRIETO ESPINOSA
ANTONIO LLORIS RUIZ
JUAN CARLOS TORRES CANTERO

McGraw Hill

INTRODUCCIÓN A LA INFORMÁTICA

CUARTA EDICIÓN

INTRODUCCIÓN A LA INFORMÁTICA

CUARTA EDICIÓN

Alberto PRIETO ESPINOSA
Catedrático de Arquitectura y Tecnología de Computadores
Universidad de Granada

Antonio LLORIS RUIZ
Catedrático de Electrónica
Universidad de Granada

Juan Carlos TORRES CANTERO
Profesor Titular de Lenguajes y Sistemas Informáticos
Universidad de Granada

Mc Graw Hill

MADRID • BOGOTÁ • BUENOS AIRES • CARACAS • GUATEMALA • LISBOA • MÉXICO
NUEVA YORK • PANAMÁ • SAN JUAN • SANTIAGO • SÃO PAULO
AUCKLAND • HAMBURGO • LONDRES • MILÁN • MONTREAL • NUEVA DELHI • PARÍS
SAN FRANCISCO • SIDNEY • SINGAPUR • ST. LOUIS • TOKIO • TORONTO

INTRODUCCIÓN A LA INFORMÁTICA. CUARTA EDICIÓN

McGRAW-HILL/INTERAMERICANA DE ESPAÑA, S. A. U.
Edificio Valrealty, 1.ª planta
Basauri, 17
28023 Aravaca (Madrid)

ISBN: 978-84-481-8532-9

Editor: Carmelo Sánchez González
Composición: Gesbiblo, S. L.

Dedicado a:

Mi madre

Alberto Prieto

Fina, María José, Fuensanta y Antonio Manuel

Antonio Lloris

Ana, Pablo, Marina, Marisa y Julia

Juan Carlos Torres

Prólogo

Contexto

Es bien conocido que la informática, a pesar de su relativa juventud (nació a mediados de los años cuarenta) es uno de los campos de mayor importancia de la actualidad, ya que es de aplicación en prácticamente la totalidad de la actividad humana, mejorando nuestra calidad de vida y el rendimiento en los procesos productivos. Una de las características más sobresalientes de la informática es que está posibilitando notablemente el avance de todas las ciencias, siendo uno de los ejemplos recientes la secuenciación del cromosoma de la especie humana, cosa imposible sin el uso de ordenadores. La penetración de la informática como herramienta en la sociedad sigue incrementándose ya que poco a poco sectores que habían estado al margen de las nuevas tecnologías están comenzando a hacer un uso intensivo de ellas y, por otra parte, los ordenadores personales han pasado a ser un componente más de nuestros hogares.

La informática tiene una serie de características que hay que tener en cuenta a la hora de desarrollar un texto general sobre ella. Entre otras se encuentran:

- Representa, sin duda, uno de los campos de la actividad humana que ha sufrido un mayor desarrollo en los últimos años.
- El sujeto principal de la informática es el computador, existiendo en la actualidad desde supercomputadores que cuestan cientos de millones de euros, hasta microcomputadores de unos pocos euros integradas en un único chip y embebidos en multitud de sistemas (electrodomésticos, redes de comunicaciones, automóviles, etc.).
- La informática puede considerarse simultáneamente como Ciencia y como Ingeniería, utilizando las metodologías tanto del desarrollo *teórico* (con las fases de definición, teorema, demostración e interpretación de resultados), como del desarrollo *experimental* (hipótesis, construcción de un modelo predictivo, diseño y realización de experimentos, y análisis de resultados) y del desarrollo tecnológico (requisitos, especificaciones, diseño, construcción del sistema, validación y prueba).

El crecimiento vertiginoso de la Informática nos ha obligado a hacer una profunda revisión en la mayoría de los capítulos de la presente edición respecto de la anterior. Por otra parte, a lo largo del texto hacemos hincapié en que los procedimientos y técnicas presentados se aplican no sólo a los computadores de uso general sino también a los de uso específico, y particularmente a los sistemas embebidos. La última característica, la informática como Ciencia e Ingeniería, subyace a lo largo de todo el libro, aunque la naturaleza de cada uno de los temas tratados en cada capítulo hace que su participación en cada una de las tres metodologías citadas (teoría, experimentación o tecnología) sea desigual.

Objetivos del libro

El objetivo de este libro es presentar una introducción a la informática, tratando tanto el hardware como el software. Para entender y tener un conocimiento profundo de la informática es necesario mostrar una visión global de ella, que permita ver cómo sus diferentes partes encajan en un todo.

Nuestra obra trata de dar una visión general de los aspectos principales de la informática, tan clara y completa como es posible. Pretendemos que la obra sea útil:

1) Dentro del contexto de una asignatura a nivel universitario de *Introducción a la Informática* (o *Informática General* o *Introducción a los Computadores* o *Fundamentos de Computadores*), en titulaciones tanto de Informática como en otras en las que se suele incluir esta materia (Ingenierías, Físicas, Matemáticas, Químicas, Ciencias Empresariales, Estadística, etc.). Debido a la autonomía universitaria, no es frecuente que este tipo de asignatura tenga un porcentaje elevado de contenidos comunes en distintos centros; ello nos ha obligado a planificar la obra de tal manera que sea posible adaptarla para ser utilizada en diversas asignaturas y contextos, sin más que alterar el orden de exposición o eliminar ciertos capítulos para explicar con más detenimiento otros.
2) Para los profesionales de la informática que deseen tener una visión actualizada de los tópicos más básicos de esta disciplina.
3) Para los usuarios de computadores que no se conformen con tan sólo saber utilizar un paquete de aplicación o un lenguaje de programación, sino que además deseen conocer los fundamentos de la máquina que ejecuta sus programas.
4) Para cualquier persona que de forma autodidacta (sin asistir a clase) tenga interés en adquirir una formación rigurosa en los aspectos más destacados de la informática.

Organización

De acuerdo con los objetivos anteriores se ha procurado que los capítulos sean lo más autónomos posibles, por lo que a veces se repiten conceptos, hecho que, por otra parte, refuerza el sentido didáctico del texto, que es un objetivo primordial en una obra como la que pretendemos.

La obra trata de dar al lector unos conocimientos básicos y terminológicos, intentando aportar una visión panorámica y equilibrada sobre todos los elementos importantes de la Informática. Para ello sus contenidos los hemos agrupado en los siguientes capítulos:

- Definiciones y conceptos básicos.
- Estructuras de datos y algoritmos.
- Representación de datos en un computador.
- Sistemas digitales combinacionales y secuenciales.
- Esquema de funcionamiento de un computador.
- Lenguajes máquina y ensamblador.
- El computador a nivel de micromáquina (diseño de un computador básico).
- El sistema de memoria.
- Periféricos de entrada/salida.
- Estructura y arquitectura de computadores.
- Sistemas Operativos.
- Conceptos básicos sobre lenguajes de programación y traductores.
- Archivos de datos.
- Bases de datos.
- El concepto de ingeniería del software.
- Transmisión y comunicación de datos.

- Redes de computadores e Internet.
- Desarrollo y evolución histórica de la Informática.

Consideraciones didácticas

En todo momento hemos tratado de enfocar los conceptos que presentamos de la forma más descriptiva y práctica posible. Uno de nuestros objetivos es hacer hincapié más en los conceptos que en los detalles, ya que de no hacerlo así, los conocimientos que pretendemos transmitir rápidamente dejarían de ser válidos. A pesar de la gran diversidad y ritmo de cambio en el campo de los computadores, esperamos haber tenido la habilidad suficiente para haber identificado los conceptos fundamentales que persistirán en el tiempo. En definitiva, pretendemos que el estudiante comprenda satisfactoriamente, además de las versiones actuales, las futuras.

La herramienta fundamental de la informática es el computador, y obviamente a él se le da una especial atención en el libro (un título alternativo podría haber sido *Introducción a los computadores*). Un computador posiblemente sea uno de los sistemas más complejos ideado por el hombre, lo que dificulta considerablemente nuestra pretensión de realizar un texto comprensible incluso estudiándolo de forma autodidacta. Para conseguir este objetivo tratamos de presentar las ideas y versiones de la forma más sencilla posible. Así, y como es habitual en las asignaturas de introducción a la informática, presentamos en distintos capítulos un computador sencillo, que denominamos CODE-2, que es una versión nueva del Ordenador Didáctico Elemental (ODE), descrito en la primera y segunda edición de este texto. Sobre CODE-2 explicamos: cómo funciona (Capítulo 7), cómo se usa (Capítulo 8) y cómo se implementa (Capítulo 9). Obviamente hemos concebido este computador con un sencillo repertorio de instrucciones (16) fácil de comprender y fácil de implementar. CODE-2 además tiene la peculiaridad de que se ha diseñado completamente utilizando distintas metodologías, y se ha construido físicamente. Además hemos desarrollado, en colaboración con alumnos de distintas promociones, emuladores y ensambladores cruzados del mismo, que están a disposición libre de los lectores en la web del texto.

Al ser los sistemas que pretendemos analizar o diseñar de gran complejidad, utilizamos ampliamente la metodología usual de la ingeniería, consistente en ver los mismos según distintos niveles conceptuales de complejidad. La idea que subyace en esta sistemática es comprender las cosas yendo *de lo sencillo a lo complejo*. Así, describimos cómo se utiliza esta metodología en el diseño de algoritmos complejos (Capítulo 3), y la computadora la presentamos según seis niveles conceptuales (Sección 1.6): desde el de máquina simbólica hasta el de lógica digital, y presentamos la descripción de redes de computadoras utilizando esta misma sistemática (Modelo OSI, Capítulo 13).

Hemos tratado de que la organización del libro sea lo más didáctica posible, para lo cual cada capitulo contiene los siguientes elementos:

- Una introducción, incluida en un cuadro, que informa del contenido del capítulo y lo enmarca dentro del contexto global de la obra.
- El texto se estructura en secciones jerárquicas (de hasta cuatro niveles), indicándose siempre al inicio de cada sección el objetivo de la misma y su estructura en secciones de nivel inferior.
- Cada capítulo incluye gran cantidad de figuras que tratan de facilitar al lector la comprensión de los conceptos que se presentan. En total el texto contiene 367 figuras.
- También se insertan multitud de tablas (149 en total) que tratan, o bien de presentar sucintamente características comparativas entre elementos de la misma naturaleza, o bien

clasificaciones o esquemas que enumeran distintas perspectivas bajo las que analizar un tema concreto.

- Para reforzar muchos conceptos, insertamos ejemplos (hasta 212) que los aplican a situaciones prácticas. Estos ejemplos se diferencian del texto principal por estar dentro de unos recuadros y con distinto tipo de letra.
- El texto principal de cada capítulo finaliza con una sección de conclusiones, que sirve para resaltar los temas abordados en el capítulo, y enlazarlo con los capítulos siguientes.
- Después, se proponen una serie de ejercicios para que el alumno pueda aplicar a casos prácticos los conceptos desarrollados en el capítulo presente y anteriores. Consideramos fundamental el refuerzo del aprendizaje con la resolución de este tipo de problemas.
- Todos los capítulos concluyen con una bibliografía, a donde el lector puede acudir para ampliar las materias correspondientes. En primer lugar las referencias bibliográficas se refieren a libros o artículos de revistas especializadas. Al ser este libro de introducción, cada capítulo podría contener una gran cantidad de referencias bibliográficas; no obstante, consideramos que para el lector es más útil disponer de un conjunto de referencias reducido, convenientemente seleccionado y actualizado. También hemos tenido especial cuidado en incluir, cuando existen, las versiones en español de las distintas obras.

Como complemento al presente texto se ha editado el libro: Prieto, A.; Prieto, B.; *Conceptos de Informática*, Serie Schaum, McGraw-Hill, 2005, que en cierta medida es un resumen de aquel complementado con multitud de problemas (487) tanto completamente resueltos (219) como sólo planteados; e incluye también ejercicios de test de elección múltiple (560 preguntas).

Servicio web

Para la presente edición del libro se ha diseñado una página web para uso de estudiantes y profesores. La dirección de acceso es:

http://www.mhe.es/universidad/informatica/introduccion

A través de esta página se pretende facilitar al lector material suplementario, tal como:

- Para alumnos y profesores:
  - Biografías de los autores.
  - Los contenidos e introducción de cada capítulo.
  - Tablas de características actualizadas.
  - Problemas y ejercicios adicionales a los que se incluyen en el texto.
  - Emulador de CODE-2.
  - Ensamblador cruzado de CODE-2.
  - Enlaces a otras páginas de interés, clasificados por temas.
  - Fe de erratas.
- Para profesores:
  - Presentaciones en PowerPoint de los distintos capítulos para impartir clases, en formato PDF (Adobe Acrobat).

Deseamos que esta página web sea un medio vivo por estar actualizada con regularidad y recoger las sugerencias de los lectores de la obra. Serán bienvenidas todas las sugerencias y comentarios tendentes a mejorar el libro.

Novedades de la cuarta edición

La cuarta edición de esta obra mejora la estructura de capítulos de la anterior y además incluye los cambios y novedades más significativos que se han producido en los cuatro años que han transcurrido desde la pasada edición. El material incluido en cada uno de los capítulos se ha revisado minuciosamente, incluyendo nuevos conceptos y eliminando los que han dejado de tener interés, actualizando tablas de características, mejorando el contenido y calidad de las figuras, y procurando dar una visión más práctica y aplicada de los distintos temas. En resumen, se han efectuado las siguientes modificaciones:

- Se ha actualizado toda la información de tipo técnico que aparece en la obra, ya que en estos últimos cuatro años, desde que se editó la edición anterior, ha habido cambios vertiginosos en la informática.
- Se han ampliado algunos conceptos, como pueden ser los de evaluación de prestaciones, sistemas embebidos, paralelismo a nivel de instrucciones y a nivel de procesador, jerarquía de memoria, memoria interna, memoria caché, etc., y se han incluido otros nuevos como los de dispositivos de memoria flash USB, unidad de control segmentada, redes de área personal, etc.
- Se ha puesto al día la bibliografía incluida al final de cada capítulo, incluyendo a las últimas ediciones de libros de texto, algunos de los cuales pueden considerarse ya clásicos en la enseñanza superior de la informática. Siempre que es posible se citan las versiones en lengua española, en caso de existir éstas como traducción de la edición más reciente de la lengua original.
- El libro se ha estructurado en capítulos más cortos, haciéndolo así más pedagógico, y pudiéndose adaptar mejor a distintos tipos de cursos, en diferentes titulaciones. De catorce capítulos y tres apéndices de la edición anterior se pasa a veinte capítulos y cuatro apéndices.
- Se ha mejorado el orden de presentación de los distintos temas en capítulos y se han agrupado algunos que aparecían dispersos en ediciones anteriores en varios de ellos. Este es el caso, por ejemplo, del capítulo dedicado al Sistema de Memoria, donde se ha incluido conjuntamente la jerarquía de memoria, la memoria interna (caché y memoria principal) y la memoria externa (discos magnéticos, cintas magnéticas, discos ópticos y memorias flash USB).
- Se incrementa el número de figuras y ejemplos insertados en el texto a lo largo de las distintas explicaciones. La mayoría de los ejemplos se corresponden con casos prácticos reales.
- El libro contiene 367 figuras, 149 tablas, 212 ejemplos y 313 ejercicios planteados.
- Se ha mejorado el formato de presentación haciéndolo más agradable y cómodo; así, los ejemplos se incluyen en recuadros para diferenciarlos de la exposición de conceptos.

Agradecimientos

Queremos agradecer la valiosa colaboración de nuestros alumnos, cuyas críticas y sugerencias han sido fundamentales. También deseamos agradecer la colaboración de diversos compañeros de los departamentos de Arquitectura y Tecnología de Computadores, de Lenguajes y Sistemas Informáticos, y de Ciencias de la Computación e Inteligencia Artificial, de la Universidad de Granada, que nos han dado valiosas sugerencias o han revisado el texto o proporcionado

nuevos problemas. Caben destacar las aportaciones de Ana Anaya, José Luis Bernier, Antonio Cañas, Antonio Díaz, Francisco Javier Fernández, Francisco Gómez Mula, Juan Manuel López Soler, Julio Ortega, Francisco J. Pelayo, David Palomar, Antonio Prados, Begoña del Pino, Beatriz Prieto, Carlos G. Puntonet, Ignacio Rojas, Fernando Rojas, y Eduardo Ros; así como la de Mancia Anguita, Pedro Castillo, Buenaventura Clares, Miguel Damas, Jesús González, Luís Javier Herrera, Francisco Illeras, Pedro Martín Smith, Eva Martínez, Juan Julián Merelo, Gonzalo Olivares, Héctor Pomares, Ignacio Requena, Manuel Rodríguez, Gustavo Romero y Moisés Salmerón. Diversos alumnos de Ingeniería Informática, Ingeniería Electrónica e Ingeniería de Telecomunicación han diseñado distintas implementaciones, simuladores y ensambladores de CODE-2, algunas de las cuales se encuentran descritas en la web del libro; a todos ellos nuestro agradecimiento.

Nuestro gratitud a los equipos de edición y producción de McGraw-Hill en Madrid, y especialmente a Concepción Fernández Madrid, que inició la edición de este libro, por su gran profesionalidad e interés en que esta cuarta edición haya visto la luz, y a Carmelo Sánchez González que la sustituyó en esta labor.

Por último, debemos mencionar que la mayoría de las horas dedicadas a la realización de este libro han sido substraídas a nuestras familias a las que agradecemos su comprensión y apoyo.

Los autores

Granada, 2 de abril de 2006

Contenido

CAPÍTULO 1

Introducción

La informática trata de la adquisición, representación, almacenamiento, tratamiento y transmisión de la información. Estas operaciones se pueden realizar automáticamente utilizando sistemas (máquinas) denominados computadores.

En este capítulo se incluyen un conjunto de conceptos que ayudan a contestar las siguientes preguntas sobre los computadores: ¿qué son?, ¿cómo se utilizan? y ¿para qué sirven?

Para ello, se dan unas definiciones y unas nociones generales sobre informática, la mayoría de las cuales son ampliadas en los siguientes capítulos.

Se pretende que el lector tenga una visión panorámica del contenido de esta obra, de forma tal que, cuando se adentre en los sucesivos capítulos, sepa enmarcar el sentido e importancia de cada uno de ellos.

1.1 Definiciones básicas

En este apartado se incluyen unas definiciones que precisan conceptos tales como informática, computador, programa, dato, instrucción y codificación de la información.

Informática es una palabra de origen francés[1] formada por la contracción de los vocablos **INFOR**mación y auto**MÁTICA**. La Real Academia Española de la lengua define la **informática** como *el conjunto de conocimientos científicos y técnicas que hacen posible el tratamiento automático de la información por medio de ordenadores.*

El término *información* hace referencia aquí a la yuxtaposición de símbolos con los que se representan convencionalmente hechos, objetos o ideas.

La informática, como disciplina, avanza gracias a la utilización de las metodologías seguidas para los desarrollos de tipo teórico y de tipo experimental, así como para el diseño de sistemas, por lo que puede considerarse tanto una ciencia como una ingeniería. La **disciplina de informática** *es el cuerpo de conocimiento que trata del diseño, análisis, implementación, eficiencia y aplicación de procesos que transforman la información* [Tuk94].

La palabra *informática* suele utilizarse como sinónimo de **ciencia e ingeniería de los computadores** (en inglés, *Computer Science and Engineering*), utilizándose habitualmente esta última terminología en los países anglosajones.

Computador, **computadora** u **ordenador** *es una máquina capaz de aceptar unos datos de entrada, efectuar con ellos operaciones lógicas y aritméticas, y proporcionar la información resultante a través de un medio de salida; todo ello sin intervención de un operador humano y bajo el control de un programa de instrucciones previamente almacenado en el propio computador.*

Se entiende por operaciones lógicas funciones tales como comparar, ordenar, seleccionar o copiar símbolos, ya sean numéricos o no numéricos.

---

[1] La definición para *Informatique* dada por la Academia Francesa es la de "Ciencia del tratamiento racional y automático de la información, considerando ésta como soporte de los conocimientos humanos y de las comunicaciones, en los campos técnicos, económico y social".

Un computador puede considerarse como un sistema (Figura 1.1) cuyas salidas o resultados son función (dependen) de sus entradas, constituidas por datos e instrucciones.

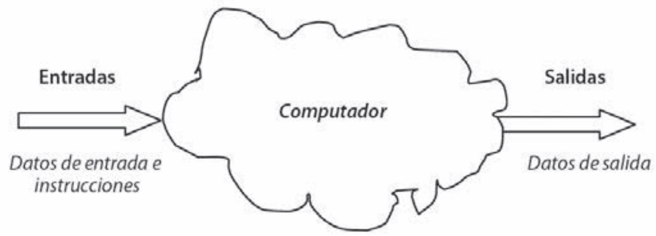


Figura 1.1. *El computador como sistema que actúa con su exterior.*

Calculadora, en su acepción actual, *es una máquina capaz de efectuar operaciones aritméticas bajo el control directo del usuario*. Es decir, calculadora no es sinónimo de computador, ya que la primera no enlaza automáticamente las operaciones que realiza, objetivo que se cubre con el programa almacenado en el caso del segundo. La calculadora tampoco realiza operaciones de tipo lógico, y sólo actúa con datos numéricos.

Considerando la definición de computador, se puede decir que **informática** o **ciencia e ingeniería de los computadores** *es el campo de conocimiento que abarca todos los aspectos del diseño y uso de los computadores*.

En la definición de computador se ha utilizado el término *datos*, que es conveniente precisar. Un **dato** *es un conjunto de símbolos utilizado para expresar o representar un valor numérico, un hecho, un objeto o una idea; en la forma adecuada para ser objeto de tratamiento.* Es decir, en informática el concepto de dato es mucho más amplio que el que se suele utilizar en Física o en Matemáticas. No sólo es dato una temperatura (25ºC), una altura (38.5m.), o una medida experimental, sino que también lo es, en informática, una matrícula de coche (7784 BBZ), el nombre de un individuo (Laura Núñez Prieto) o una frase de un libro.

Los datos pueden ser captados directamente por el computador (por ejemplo, detectando electrónicamente un sonido o fonema, una temperatura, la silueta de una figura, o el paso de un objeto), o pueden ser dados en forma de letras y números (grafismos).

Los grafismos (caracteres) resultan muy útiles, dada la gran variedad de informaciones que con ellos se puedan representar. De hecho esta es una de las formas más habituales de transmitir, comunicar o almacenar información en la sociedad actual: el lenguaje escrito.

Los grafismos utilizados normalmente en informática son los caracteres numéricos (las diez cifras decimales), los caracteres alfabéticos y los caracteres especiales (símbolos ortográficos, aritméticos y otros). Cualquier información (datos o instrucciones) puede expresarse utilizando caracteres, y así ser introducida en el computador. De igual forma, usualmente el computador nos proporciona los resultados en forma escrita, utilizando caracteres.

Frecuentemente las salidas de un programa (como, por ejemplo, los resultados de un cálculo matemático, o de la búsqueda de un nombre), se denominan también datos (**datos de salida**), pudiendo eventualmente utilizarse estos como datos de un programa posterior. Es decir, la palabra *dato* se utiliza como contraposición a *instrucción*. El computador actúa con dos tipos de informaciones: instrucciones (que indican a la máquina qué es lo que tiene que hacer) y datos (que son los elementos que procesa o genera el programa).

En informática es frecuente codificar la información. **Codificación** *es una transformación que representa los elementos de un conjunto mediante los de otro, de forma tal que a cada elemento del primer conjunto le corresponda un elemento distinto del segundo*.

Ejemplos de códigos son:

- El código postal asociado a los distritos postales de un estado.
- El código de enfermedades definido por la Organización Mundial de la Salud (OMS). A cada enfermedad se le asigna un código.
- El número de un carné de identidad. A cada persona se le asocia un número, pudiendo referirse administrativamente a ella por medio de ese código.

Con los códigos se puede comprimir y estructurar la información. La identificación de un coche por su matrícula (que es un código más) es más corta que hacerlo, por ejemplo, por el nombre de su propietario, su marca, color y fecha de compra.

Pueden definirse códigos con *significado*, así los códigos postales no se dan al azar, sino que, por ejemplo en la Unión Europea, la letra inicial corresponde al estado (E, España), las dos primeras cifras a la provincia (por ejemplo, 18 Granada) y el resto de las cifras al distrito postal, dentro de la provincia.

Como se verá en la Sección 4.2, en el interior de los computadores la información se almacena y se transfiere de un sitio a otro según un código que utiliza sólo dos valores (un *código binario)* representados por *0* y *1* (Figura 1.2). En la entrada y salida del computador se efectúan automáticamente los cambios de código oportunos para que en su exterior la información sea directamente comprendida por los usuarios. Todos los aspectos relativos a la representación de la información, tanto en el exterior como en el interior del computador se detallan en el Capítulo 4.

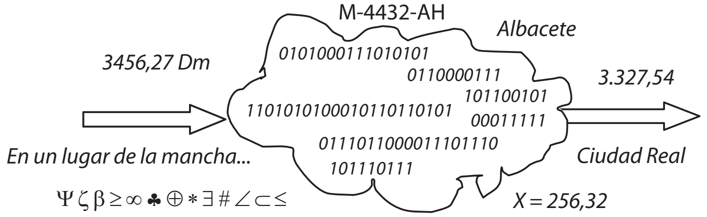


Figura 1.2. *En el interior del computador todos los símbolos externos se representan con ceros y unos.*

La unidad más elemental de información es un valor binario, conocido como *bit*. El origen de este término es inglés, y se suele considerar que procede de la contracción de las palabras **BI**nary y digi**T**. Un **bit** es, por tanto, *una posición o variable que toma el valor 0 ó 1.* Obviamente la capacidad mínima de almacenamiento de información en el interior de un computador es el bit, pudiéndose medir la capacidad de memoria de un computador en bits (en la práctica no se hace así, por ser el bit una unidad excesivamente pequeña). El bit representa la información correspondiente a la ocurrencia de un suceso de entre dos posibilidades distintas. Por ejemplo, un bit es la cantidad de información correspondiente a un mensaje anunciando si determinado caballo ha ganado (1) o no (0) una carrera.

Anteriormente se indicó que la información se representa por medio de caracteres y que internamente se codifica en un alfabeto binario; es decir, en bits. Por tanto a cada carácter le corresponde cierto número de bits. Un **byte** *es un conjunto de 8 bits considerado como una unidad.* Tradicionalmente un byte era el *número de bits necesarios para almacenar un carácter*, pero en la actualidad se considera como sinónimo de grupo de 8 bits u octeto. La capacidad de almacenamiento de un computador o de un soporte de información (tal como un disco o cinta magnética) se suele medir en bytes. Como el byte es una unidad relativamente pequeña, es usual utilizar los múltiplos que se indican en la Tabla 1.1, que son similares a los utilizados en física, pero con la diferencia de que son potencias enteras de 2, o tomando como punto de partida el kilo, potencias de 1.024 en lugar de potencias de 1.000.

Tabla 1.1. *Múltiplos y submúltiplos establecidos para medidas por el SI (Système Internacional d'Unités).*

MÚLTIPLOS			
Prefijo	**Símbolo**	**Factor decimal**	**Factor binario (Nº de Bytes o bits)**
Kilo-	k-	$1000^1 = 10^3$	$1024^1 = 2^{10} = 1.024$
Mega-	M-	$1000^2 = 10^6$	$1024^2 = 2^{20} = 1_1048.576$
Giga-	G-	$1000^3 = 10^9$	$1024^3 = 2^{30} = 1.073_1741.824$
Tera-	T-	$1000^4 = 10^{12}$	$1024^4 = 2^{40} = 1_2099.511_1627.776$
Peta-	P-	$1000^5 = 10^{15}$	$1024^5 = 2^{50} = 1.125_2899.906_1842.624$
Exa-	E-	$1000^6 = 10^{18}$	$1024^6 = 2^{60} = 1_3152.921_2504.606_1876.976$
Zetta-	Z-	$1000^7 = 10^{21}$	$1024^7 = 2^{70} = 1.180_3591.620_2717.411_1303.424$
Yotta-	Y-	$1000^8 = 10^{24}$	$1024^8 = 2^{80} = 1_4208.925_3819.614_2629.174_1706.176$

Los prefijos (k-, M-, G-, T-, P- y E-) no sólo se utilizan con Bytes, sino también con otras unidades internas de información. Así, 1 Gb (o Gigabit) son $1.073_1741.824$ bits.

En general, utilizaremos de aquí en adelante una "b" (minúscula) para indicar bit y una "B" (mayúscula) para indicar byte.

1.2 Estructura funcional de los computadores

En la Figura 1.3 puede verse un esquema general de un computador sencillo, que se compone de los siguientes elementos o **unidades funcionales**: unidades de entrada, unidades de salida, memoria interna, memoria externa, unidad aritmético-lógica y unidad de control. Este diagrama corresponde a los primeros computadores, denominados **computadores von Neumann** (véase Sección 20.2), pero sigue siendo conceptualmente válido hoy día. A continuación se describen brevemente las distintas unidades funcionales, que serán estudiadas con más detalle en los capítulos del 7 al 12 del presente texto.

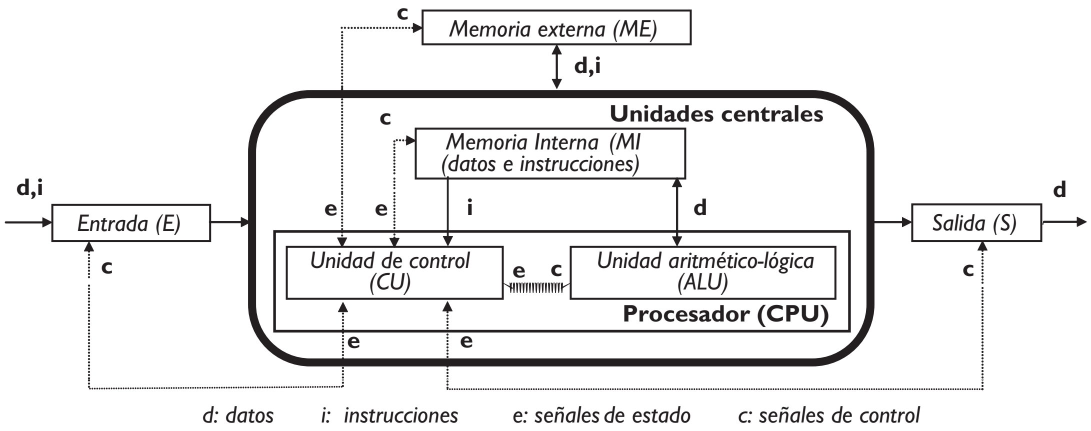


Figura 1.3. *Esquema que muestra las unidades funcionales de un computador.*

Unidad de entrada (E)
Es un dispositivo por el que se introducen en el computador los datos e instrucciones. En estas unidades se transforman las informaciones de entrada en señales binarias de naturaleza eléctrica. Un mismo computador puede tener distintas unidades de entrada (en la Figura 1.3 se incluye una sola por simplificar). Son unidades de entrada: un teclado, un ratón, un escáner de imágenes, una lectora de tarjetas de crédito, etc.

Unidad de salida (S)
Es un dispositivo por el que se obtienen los resultados de los programas ejecutados en el computador. La mayor parte de estas unidades (un computador suele tener varias de ellas) transforman las señales eléctricas binarias en información perceptible por el usuario. Son dispositivos de salida unidades tales como una pantalla, una impresora o un altavoz.

Memoria interna (MI)
Es la unidad donde se almacenan tanto los datos como las instrucciones durante la ejecución de los programas. La memoria interna (también a veces denominada **memoria central** o **memoria principal**) actúa con una gran velocidad y está ligada directamente a las unidades más rápidas del computador (unidad de control y unidad aritmético-lógica). Para que un programa se ejecute debe estar almacenado (*cargado*) en la memoria principal. En los computadores actuales está formada por circuitos electrónicos integrados (chips).

La memoria está dividida en ***posiciones*** (denominadas también ***palabras de memoria***) de un determinado número de bits *n* (Figura 1.4), que es donde se almacena o memoriza la información. Cada palabra únicamente se puede referenciar por su dirección (número de orden), de forma que siempre que se quiera escribir o leer un dato o instrucción en la memoria hay que especificar la dirección dónde se debe efectuar la operación en cuestión. En la Figura 1.4 la información 48AB está almacenada en la posición 3; cuando se desee leer este dato, la unidad de control debe dar la orden de leer el contenido de la posición (dirección) 3.

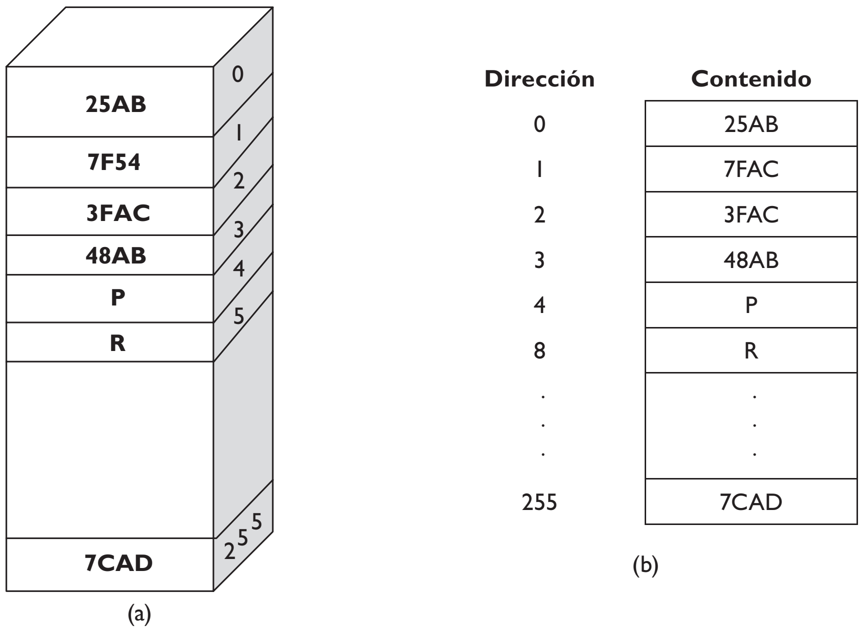


Figura 1.4. *Diagrama simplificado de una memoria principal de 255 palabras (en realidad tanto las direcciones como los contenidos se representan con códigos binarios).*

Normalmente hay una zona de la memoria en la que sólo se puede leer (**memoria ROM**) y que es permanente (al desconectar el computador su información no se pierde), y otra en la que se puede leer y escribir (**memoria RAM**) y que es volátil. La memoria ROM de los computadores viene grabada de fábrica, y contiene programas y datos relevantes del sistema operativo que deben permanecer constantemente en la memoria interna.

Memoria Externa (ME)
La memoria interna es muy rápida (es normal que pueda leer o escribir decenas de millones de palabras en un solo segundo), pero no tiene gran capacidad para almacenar información y es su zona RAM es volátil. Para guardar masivamente información se utilizan otros tipos de memoria, tales como discos magnéticos, discos ópticos y cintas magnéticas, que son más lentos pero pueden tener mucha más capacidad que la memoria principal (del orden de un millón de veces más lentos y de mil veces más capaces, en el caso de un disco magnético). El conjunto de estas unidades se denomina **memoria externa, memoria masiva, memoria auxiliar** o **memoria secundaria**. Usualmente los datos y programas se graban (introduciéndolos por las unidades de entrada) en la memoria externa, de esta forma cuando se ejecute varias veces un programa o unos datos se utilicen repetidamente, no es necesario darlos de nuevo a través del dispositivo de entrada. La información guardada en un disco o cinta permanece indefinidamente hasta que el usuario expresamente la borre.

Unidad de tratamiento (PU, *Proccesing Unit*)
Como elemento principal contiene a la unidad aritmético-lógica o **ALU** (*Arithmetic Logic Unit)*, contiene los circuitos electrónicos con los que se hacen las operaciones de tipo aritmético (sumas, restas, etc.) y de tipo lógico (comparar dos números, hacer operaciones del álgebra de boole binaria, etc.). Esta unidad también suele denominarse **camino de datos** (o **ruta de datos**) ya que aparte de contener a la ALU incluye otros elementos auxiliares por donde se transmiten (**buses de datos**), o registros para almacenar temporalmente los datos al objeto de operar con ellos. Un **registro** es una pequeña memoria diseñada para almacenar un dato, instrucción o dirección de memoria.

Unidad de control (CU, *Control Unit*)
La unidad de control detecta *señales eléctricas de estado* procedentes de las distintas unidades, indicando su situación o condición de funcionamiento (Figura 1.3). También capta secuencialmente de la memoria las instrucciones del programa, y, de acuerdo con el código de operación de la instrucción captada y con las **señales de estado** procedentes de los distintos elementos del computador, genera **señales de control** dirigidas a todas las unidades, ordenando las operaciones que implican la ejecución de la instrucción. La unidad de control contiene un **reloj**, que sencillamente es un *generador electrónico de pulsos* que sincroniza todas las operaciones elementales del computador. El período de esta señal se denomina **tiempo de ciclo**, y está comprendido aproximadamente entre décimas de nanosegundos y varios microsegundos, dependiendo del computador. La **frecuencia del reloj** (inverso del tiempo de ciclo) suele darse en millones de ciclos/segundo, (Megahercios o, abreviadamente, **MHz**) o en miles de millones de ciclos/segundo (Gigahercios, GHz). La ejecución de cada instrucción supone la realización de un conjunto de operaciones elementales consumiendo un número predeterminado de ciclos, de forma que las instrucciones más complejas utilizan un número mayor de ciclos que las menos complejas.

Unas unidades están interconectadas con otras según se indica en la Figura 1.3; ahora bien, existen diversas variantes a este esquema, dependiendo de la estructura o configuración concreta del computador, según se analizará en el Capítulo 12. Los distintos elementos de un

computador se interconectan a través de conjuntos de hilos, líneas o pistas eléctricamente conductores que suelen llevar en un instante dado *(en paralelo)* la información completa de una instrucción, un dato o una dirección. Un conjunto de conductores que transmite información del mismo tipo entre unidades distintas se denomina **bus**. El **ancho de un bus** es el número de hilos que contiene, o número de bits que transmite simultáneamente, en paralelo.

Un computador actual puede tener gran cantidad (cientos) de unidades de entrada o salida. Piénsese, por ejemplo, en los terminales y cajeros de tarjetas de crédito (que son dispositivos de E/S) de una entidad bancaria o de una empresa de transporte aéreo. La conexión de las unidades de E/S puede hacerse directamente con un bus, o a través de una línea telefónica o de un enlace de radio o con fibra óptica (véase Capítulo 18).

Se denominan **periféricos** de un computador al conjunto de sus unidades de E/S y de memoria externa. Al resto de unidades; es decir, memoria interna, y unidades de control y ALU (que figuran en un recuadro en la Figura 1.3) las denominaremos en este y sucesivos capítulos, **unidades centrales**. En la Figura 1.5 se muestra una vista general de un computador tipo PC con distintos periféricos de entrada (teclado, ratón y escáner de imágenes), de salida (monitor de vídeo, impresora y altavoces) y dispositivos de memoria masiva (unidad de disco, unidad de disquete y unidad CD/DVD). La **unidad de procesamiento central** (**CPU**, *Central Processing Unit*) o, sencillamente **procesador** es el conjunto de unidad de control y unidad de tratamiento. En la Tabla 1.2 se resumen estos conceptos.

Tabla 1.2. *Clasificación de las unidades funcionales de un computador.*

Unidades centrales	– Procesador o unidad central de procesamiento (CPU)	• Unidad de tratamiento o camino de datos (ALU)
		• Unidad de Control (CU)
	– Memoria interna (MI)	
Periféricos	– Memoria externa (ME)	
	– Dispositivos de entrada (E)	
	– Dispositivos de salida (S)	

El grado de miniaturización alcanzado en la integración de circuitos electrónicos ha llegado a ser tan alto que en un único chip se pueden incluir todos los elementos de un procesador. Un **microprocesador** *es un procesador (CPU) implantado en un circuito integrado (o en un conjunto muy reducido de ellos).* Un microprocesador por sí solo no realiza ninguna función; para funcionar adecuadamente debe estar interconectado a un conjunto de circuitos a los que controla o monitoriza, formando con estos un sistema electrónico digital programable (un computador, por ejemplo). Conviene resaltar que en el caso de los microprocesadores, el prefijo *micro* hace referencia al tamaño del procesador y no a sus prestaciones. De hecho con microprocesadores se construyen desde calculadoras de bolsillo, PC y estaciones de trabajo, hasta los supercomputadores actuales más potentes.

Un **microcontrolador** es un circuito integrado que contiene, total o parcialmente, los cinco elementos básicos de un computador completo (unidad de control, unidad de tratamiento, memoria y puertos de entrada/salida), estando proyectados para aplicaciones de supervisión, monitorización, gestión y control en sistemas tales como aparatos telefónicos, electrodomésticos, instrumentación médica, control de robots, líneas de ensamblado, etc. Se

diferencian de los microprocesadores en que: 1) contienen en su interior no sólo el procesador sino también otros elementos como puertos de entrada/salida y memoria interna (ampliable externamente), 2) están orientados a aplicaciones específicas de control, y suelen instalarse *embebidos* dentro del sistema que controlan (máquina de lavar, automóvil, máquina de juegos, impresora, etc.).



Figura 1.5. *Vista general de un PC con sus periféricos.*

De la descripción realizada hasta ahora en esta sección se deduce que un computador es un sistema complejo que está formado por distintas unidades, módulos o dispositivos ensamblados adecuadamente uno con otro. En muchos casos es necesario adaptar las características (niveles eléctricos, velocidad, etc.) de dos módulos que se acoplan, para que la conjunción de los dos funcione adecuadamente, o entre un módulo y su entorno. El conjunto de elementos adaptadores que sirven de comunicación entre dos módulos, genéricamente se denomina **interfaz**. El concepto de interfaz se aplica también a los programas, de forma que puede hablarse de la **interfaz entre dos programas**, e **interfaz de usuario** que es el conjunto de instrucciones que hace que un programa o aplicación intercambie información con el usuario del mismo.

1.3 Parámetros para la caracterización de prestaciones

Existen varias magnitudes que determinan las prestaciones de las distintas unidades que componen un computador. En relación con ellas, a continuación vamos a examinar los conceptos de capacidad de almacenamiento, tiempo de acceso, longitud de palabra, ancho de banda y

rendimiento del procesador. Otros parámetros de interés, como la productividad y tiempo de respuesta, y que dan una valoración global del sistema, considerando tanto el comportamiento de la máquina como del sistema operativo, se considerarán en la Sección 13.5.

La **capacidad de almacenamiento** se refiere a las posibilidades de una unidad para almacenar datos o instrucciones de forma temporal o fija. El procesador contiene registros de uso general, y su capacidad de almacenamiento viene dada por el número de ellos y su longitud. La capacidad de la memoria, tanto interna como externa, se da en bytes (MB, GB, TB, etc.).

El **tiempo de acceso** de una unidad de memoria es el intervalo de tiempo que transcurre desde el instante en que se proporciona a la misma la *posición* (*dirección*) concreta del dato o instrucción que se quiere leer o escribir, y el instante en que se obtiene (lee) o graba (escribe) el mismo.

Para muchas operaciones que se realizan en el computador, el byte (8 bits) es una unidad de información muy pequeña; así las ALU suelen operar con datos de mayor longitud, normalmente de un número entero de bytes, siendo valores habituales los siguientes: 8, 16, 32, 64 o 128 bits. Se denomina **palabra** al conjunto de bits que forma un dato con el que opera la ALU, y coincide, en general, con el número de bits de la mayoría de los registros del procesador. La **longitud de una palabra** es el número de bits que la forman; así, si una ALU opera con datos de 32 bits, la longitud de palabra de ese procesador es de 32 bits. Con frecuencia la longitud de la palabra coincide con el ancho del bus de datos que conecta el procesador con la memoria. También se habla de **palabra de memoria**, que es sencillamente la información que se graba en cada una de las posiciones especificadas a través del bus de direcciones. Este último término es más confuso ya que normalmente la memoria se suele organizar en módulos que actúan en paralelo y pueden escribirse o captarse datos de distinta longitud. En efecto, en la mayoría de computadores de longitud de palabra de 32 bits, el direccionamiento a memoria se efectúa por bytes, y es posible acceder directamente a bytes (8 bits), medias palabras (16 bits) y palabras (32 bits); algunos incluso permiten acceder a dobles palabras (64 bits). Esta cuestión se analizará con más detalle en el Capítulo 10.

La longitud de palabra determina de forma indirecta la velocidad del computador, ya que:

- Si en un programa se establece que los cálculos deben realizarse con una precisión de 32 bits y la longitud de palabra es de 16 bits, el número de instrucciones del programa al menos se duplica ya que todos los datos habrá que proporcionarlos (con el programa adecuado) a la ALU troceados en dos partes de 16 bits cada una de ellas, primero la menos significativa y después la más significativa.
- Cada instrucción ocupa un número determinado de bits (32, por ejemplo). Si la longitud de palabra es menor que el tamaño de las instrucciones (16 bits, por ejemplo), para que el procesador capte cada instrucción de la memoria principal tendrá que realizar varias lecturas sucesivas de la memoria (2, en el caso del ejemplo). Por otra parte, si el tamaño de las instrucciones se ajusta al de la palabra, entonces puede restringirse la variedad de las instrucciones y la capacidad máxima de memoria, ya que en la propia instrucción va codificada la identificación de la misma (código de operación) y puede ir la dirección de la posición de memoria a la que hay que acceder, en su caso.

Otro parámetro de interés ligado a la implementación del computador es el **ancho de banda**, que representa la cantidad de información transferida por segundo entre una unidad y otra. Por ejemplo, decir que el *ancho de banda entre la memoria y el procesador es de*

133 MB/s, quiere decir que en 1 segundo se pueden transferir 133 Megabytes entre las unidades citadas.

Desde el punto de vista de los usuarios interesa una medida más global del funcionamiento del computador, que pueda servir para comparar la *potencia de procesamiento* de dos equipos distintos. Dado un determinado programa, diremos que un computador tiene mayor rendimiento que otro si el primero lo ejecuta en menos tiempo. El **tiempo de ejecución** de un programa, t_E, es el tiempo que transcurre desde el inicio hasta el final de su ejecución. Si denominamos N_I al número de instrucciones que se ejecutan en un programa[2], N_{CI} al número medio de ciclos de reloj que consume cada instrucción y F a la frecuencia del reloj, el tiempo de ejecución del programa vendrá dado por:

[1.1]

$$t_E = N_I \cdot N_{CI} \cdot T = \frac{N_I \cdot N_{CI}}{F}$$

Por otra parte, el **rendimiento de un computador** en la ejecución de un programa es la inversa del tiempo de ejecución:

[1.2]

$$\eta_E = \frac{1}{t_E}$$

Uno de los objetivos básicos de la arquitectura de computadores es reducir el valor de t_E, para lo cual debe aumentar F o disminuir N_I o N_{CI}. El valor de F viene determinado por la velocidad de funcionamiento de los circuitos integrados y es responsabilidad de la tecnología electrónica; sin embargo, la disminución de N_I y N_{CI} dependen de la arquitectura del computador, aunque los tres parámetros están interrelacionados. Claramente de la expresión [1.1] se deduce que un procesador con mayor frecuencia no tiene porqué ser más rápido que otro de frecuencia menor ejecutando el mismo programa (hay que considerar también los valores de N_I y N_{CI}).

EJEMPLO 1.1

Según indicamos anteriormente, la ejecución de cada instrucción para un procesador consume un número determinado de ciclos de reloj, existiendo instrucciones más rápidas que otras. Así, por ejemplo, en el procesador Pentium III una instrucción puede consumir 14 ciclos de reloj, mientras que otra puede consumir 45. Debido a lo anterior no se puede comparar el rendimiento de dos computadores con procesadores diferentes por sus frecuencias de reloj, ya que un computador A con frecuencia de reloj menor que otro B puede que esté diseñado de forma que sus instrucciones se ejecuten consumiendo menos ciclos que el B, siendo factible en este caso, que el rendimiento del A sea mayor que el del B. En realidad, como hemos puesto de manifiesto con la expresión [1.1], el rendimiento de un procesador depende de tres factores: el repertorio de instrucciones (factor que determina N_I), la frecuencia de reloj (F) y el número de ciclos asociados a cada instrucción (N_{CI}).

[2] Este número no tiene porqué coincidir con el número de instrucciones que tiene el programa, ya que éste puede contener bucles o lazos que se ejecuten múltiples veces e instrucciones en ramas del programa que no se ejecuten, todo ello dependiendo de los datos de entrada al programa.

La **tasa de ejecución de instrucciones** (R_I), o velocidad de funcionamiento de un procesador, se suele dar en *millones de instrucciones por segundo* (**MIPS**), y se pueden expresar como:

[1.3]

$$R_I = \frac{N_I}{t_E \cdot 10^6} \; MIPS$$

donde t_E es el tiempo, en segundos, de ejecución de las N_I instrucciones.

Los computadores dedicados a aplicaciones científicas o técnicas operan fundamentalmente con números reales, cuya representación dentro del computador se conoce con el nombre de **datos de coma flotante** (Sección 4.5.2). En este caso una medida más apropiada que los MIPS para caracterizar la velocidad del procesador son los ***MFlops*** (que se lee "Mega FLOPS"; FLOPS es un acrónimo de *FLOat Point per Second*) o *millones de operaciones con datos de coma flotante por segundo*:

[1.4]

$$R_{OFloat} = \frac{N_{OFloat}}{t_E \cdot 10^6} \; Mflops$$

donde N_{OFloat} es el número de operaciones en coma flotante (sumas, restas, multiplicaciones o divisiones de números reales), t_E es el tiempo de ejecución de esas operaciones y R_{OFloat} es la tasa de ejecución de operaciones en coma flotante.

Tanto las medidas en MIPS como en MFlops por sí solas no son significativas para comparar dos computadores ya que las tasas de ejecución dependen del programa concreto que se procese y de sus datos: el hecho de que un programa se ejecute más rápidamente en un computador que en otro no implica que vaya a suceder lo mismo para cualquier otro programa (depende de las instrucciones concretas que formen cada uno de los programas). Para poder evaluar lo más correctamente posible el rendimiento de un computador y poder hacer análisis comparativos se han establecido por la comunidad informática:

a) Computadores de referencia; así desde 1977 era habitual utilizar para este cometido el VAX-11/780 y a partir de 1995 el Sun SPARC 10/40.
b) Conjuntos de programas de prueba (***benchmarks***).

Al utilizar un computador de referencia se pueden dar la tasa de ejecución de instrucciones del computador a caracterizar ("computador X") *normalizada* con respecto al computador de referencia:

[1.5]

$$R_{i,relativa} = \frac{t_{i,referencia}}{t_{i,computador\ X}} \cdot R_{i,referncia} \; MIPS \text{ o } MFLOPS$$

donde, en la expresión anterior, los tiempos se refieren a la ejecución del mismo programa *i*.

Cuando se utilizan conjuntos de programas de prueba, se hacen las medidas normalizadas [1.5] para cada uno de los programas del conjunto y el valor final para la tasa de ejecución (o velocidad) se da como media geométrica de las medidas parciales obtenidas:

[1.6]

$$R = \sqrt[n]{\prod_{i=1}^{n} R_{i,relativa}}$$

Se utiliza la media geométrica, en lugar de la aritmética, ya que la primera disminuye el efecto de los valores extremos en la medida final, y además porque la media aritmética depende de la máquina que se tome como referencia, cosa que no ocurre con la media geométrica (véase Problema 1.11).

Uno de los conjuntos de prueba más conocidos y completos es el ***SPEC*** (*System Performance Evaluation Cooperative* o conjunto para evaluación del rendimiento de computadores). El conjunto SPEC95 da como medida un valor relativo al computador Sun SPARCstation 10/40, de esta forma la medida no tiene dimensiones.

EJEMPLO 1.2

El Pentium Pro de 200 MHz tiene un valor SPEC95 de valor 8 para enteros y 7 para reales, lo que quiere decir que se considera 8 veces más rápido que el procesador Sun citado en aplicaciones que utilicen números enteros, y 7 en el caso de números reales.

En la actualidad se utiliza el SPEC CPU2000 que incluye 19 aplicaciones de prueba nuevas (compresión de datos, procesamiento de textos, juego del ajedrez, etc., véase Problema 1.11), no consideradas previamente en el conjunto SPEC95 [Hen00].

La tasa de ejecución de instrucciones SPEC, para cada programa, i, de prueba se obtiene de acuerdo con la expresión [1.5], y la tasa SPEC final del computador con la media geométrica [1.6] de las tasas de los n programas de referencia establecidos.

Cuando un computador se utiliza preferentemente para cálculo científico o técnico es común utilizar como conjunto de prueba, en lugar del SPEC, el LINPACK, que contiene programas de álgebra para resolver sistemas de ecuaciones lineales, problemas de ajustes lineales por mínimos cuadrados, etc. Este tipo de programas caracteriza muy bien a las aplicaciones científicas ya que éstas suelen utilizar pequeños programas cuyo tiempo de ejecución es muy grande.

Otros ejemplos de programas de pruebas son Dhrystone, iCOMP, Livermore, Whetstone o Ciusbet.

1.4 Programas e instrucciones

Una **instrucción** *es un conjunto de símbolos que representa una orden de operación o tratamiento para el computador*. Las operaciones suelen realizarse con datos.

Un **programa** *es un conjunto ordenado de instrucciones que se dan al computador indicándole las operaciones o tareas que se desea que realice.*

Las instrucciones se forman con elementos o símbolos tomados de un determinado repertorio y se construyen siguiendo unas reglas precisas.

Todo lo relativo a los símbolos y reglas para construir o redactar con ellos un programa se denomina **lenguaje de programación**.

Las instrucciones de un lenguaje de programación se pueden clasificar en los siguientes grupos:

- **Instrucciones de transferencias de datos.** Por ejemplo, de entrada o lectura (llevar un dato de una unidad de entrada a la memoria o a un registro del camino de datos),

de salida o escritura (llevar un dato de la memoria o de un registro a una unidad de salida, llevar un dato de la memoria a un registro o viceversa, etc.).

- **Instrucciones de tratamiento.** Por ejemplo, sumar dos datos, comparar dos datos para comprobar si son iguales, o uno mayor que otro. Aquí se incluyen las instrucciones aritmético-lógicas.
- **Instrucciones de flujo de control o de bifurcación y saltos.** Las instrucciones de un programa se ejecutan por el computador ordenadamente una tras otra (es decir, *secuencialmente*). Las instrucciones de flujo de control permiten alterar el orden de ejecución. Existen instrucciones que permiten interrumpir la ejecución de un programa y saltar a ejecutar otro programa (que genéricamente se denomina ***rutina***), cuando finaliza este último continúa ejecutándose el programa inicial en el punto donde se interrumpió.
- **Otras instrucciones.** Tal como detener el funcionamiento del computador a la espera de una acción del operador.

Los circuitos electrónicos de la unidad de control del computador sólo pueden interpretar instrucciones de un determinado lenguaje, denominado **lenguaje máquina**. Las instrucciones de este lenguaje están formadas por bits (ceros o unos) agrupados usualmente en distintos bloques o campos. Siempre hay un campo de **código de operación** (abreviadamente, **codop**) que identifica la operación correspondiente a la instrucción. Este código se obtiene de una tabla o repertorio en el que figuran las instrucciones que la máquina puede ejecutar y el código binario asociado a cada una de ellas. Suele haber otros campos, entre los que se encuentran los de direcciones. Un **campo de dirección** específica el lugar (registro o posición de memoria) dónde se encuentra un dato con el que hay que operar, o dónde hay que llevar un resultado, en consonancia con el codop.

El lenguaje máquina tiene serios inconvenientes, como son: depende del procesador; el repertorio de instrucciones es muy reducido, conteniendo sólo operaciones muy elementales; es muy laborioso programar con él por tener que utilizar sólo números; etc. (véase Capítulo 8). Para evitar estos problemas se han ideado **lenguajes de alto nivel**, que no dependen del computador, y se han proyectado pensando en facilitar la tarea de programación. Las empresas de informática suministran programas **traductores***,* que al ejecutarlos en el propio computador e introduciendo como datos programas escritos en el lenguaje de alto nivel, generan como resultado programas en lenguaje máquina (Figura 1.6). En la Figura 1.7 se muestra una instrucción en lenguaje C y su correspondiente traducción al lenguaje máquina de un Pentium. Una vez traducido un programa escrito en un lenguaje de alto nivel, puede ser ejecutado directamente por la unidad de control.

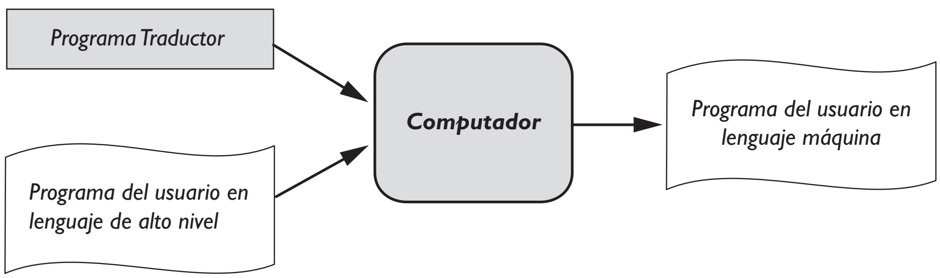


Figura 1.6. *Proceso de traducción de un programa en lenguaje de alto nivel (LAN) a lenguaje máquina (LM).*

Instrucción en C	Instrucciones en ensamblador	Instrucciones en código máquina Direcc.	Instruc.	Explicación
for (i=0;i<9000;i++) A(i)=0	mov cx,9000	0000	B9	Nº de veces a repetir la instrucción "rep stosw"
		0001	28	
		0002	23	
	les di,A	0003	C4	Se da a es:di la dirección inicial de A
		0004	3E	
		0005	00	
		0006	00	
	cld	0007	FC	Avanzar el puntero una posición
		0008	33	
	xor ax,ax	0009	C0	Hacer AX=0
	rep stosw	000A	F3	Almacenar el valor de AX en A[.]
		000B	AB	

Figura 1.7. *Instrucción en C para poner a 0 los elementos de una tabla y su correspondiente traducción a lenguaje máquina.*

Son lenguajes de alto nivel FORTRAN, COBOL, BASIC, Lisp, Prolog, Logo, Pascal, C, Ada, C++, Java, Visual Basic, etc. (Capítulo 14).

Existen dos tipos de traductores: compiladores e intérpretes. Los **compiladores** traducen el programa inicial (programa fuente) considerándolo globalmente, y generan un programa (programa objeto), como resultado de la traducción, que se almacena en un archivo en disco. Para ejecutar posteriormente el programa, se puede utilizar directamente el programa objeto. Los traductores **intérpretes**, en lugar de considerar al programa a traducir como un todo, van analizando, traduciendo y ejecutando una a una las instrucciones del programa fuente; no se analiza una instrucción hasta que la anterior se haya ejecutado. Los intérpretes no generan programa objeto almacenable en un archivo, necesitándose, siempre para ejecutar el programa, volver a traducirlo.

Además de los programas traductores, el constructor proporciona otros programas que son necesarios para el control y para la utilización eficiente y cómoda del computador. Algunos de estos programas son para editar (introducir u obtener la información en forma adecuada, y con posibilidades de modificar, insertar o borrar dicha información), para copiar un programa de disco a memoria o viceversa, o para borrar un conjunto de datos de un disco, o para comprobar el funcionamiento de los circuitos del computador (autodiagnóstico). El conjunto de programas que controlan y gestionan los recursos del computador se denomina sistema operativo. Un **sistema operativo** *es un programa compuesto por una serie de módulos o subprogramas que controlan el funcionamiento global del computador de forma que este sea utilizado por los usuarios de forma eficiente y cómoda.* Los subprogramas del sistema operativo se utilizan con un lenguaje específico denominado **lenguaje de control**. A las instrucciones del lenguaje de control se las suele denominar **órdenes** (*commands*).

Los sistemas operativos actuales están diseñados de forma que posibilitan el uso de un computador concurrentemente por varios usuarios, la distribución de los recursos del sistema dentro de redes de computadores, la utilización por los programas de una memoria principal de capacidad mayor que la memoria física real, y el uso de sofisticadas interfaces de usuarios tales como gráficos, ratón y ventanas junto con un lenguaje de control natural.

Por lo general cada constructor de computadores tiene sus propios sistemas operativos; no obstante en la actualidad sólo unos pocos se mantienen en uso tales como son el UNIX, Linux, Microsoft, Mac OS, OS/2 y AIX.

Los Capítulos 13 y 14 se dedican íntegramente a ampliar los conceptos vistos en esta sección.

1.5 Tipos de computadores

Los computadores pueden clasificarse atendiendo a muy distintos criterios, que se resumen en la Tabla 1.3.

A) Clasificación según la generalidad de su uso

Este criterio de clasificación de los computadores hace referencia al *uso* o *propósito* para el que fueron diseñados y construidos.

Computador de uso general

Es el computador que puede utilizarse para distinto tipo de aplicaciones, tales como gestión administrativa, cálculo científico o cálculos técnicos. Que realice una aplicación u otra depende del programa que el usuario ordene ejecutar.

Computador de uso específico

Es el computador que *únicamente* puede utilizarse para un grupo de aplicaciones determinado o una aplicación muy concreta. Por lo general se construyen con microcontroladores o microprocesadores. Estos computadores tienen la estructura indicada en la Sección 1.2.1, siendo válido para ellos todo lo indicado en éste y sucesivos capítulos. La mayoría de computadores de uso específico son **computadores embebidos**, que forman parte de algún sistema y que suelen realizar funciones de control pero no se puede acceder a ellos directamente, tal es el caso de los relojes de cuarzo, cámaras de fotos, grabadores de vídeo y otros muchos equipos domésticos e industriales. Otros ejemplos de computadores de uso específico son: un vídeo-juego de bolsillo, el computador que contiene un robot, el que contiene un misil para guiar su trayectoria o un computador para control de tráfico

Tabla 1.3. *Criterios para clasificación de los computadores.*

Criterio	Tipos
Generalidades de uso	– Uso general – Uso específico • Sistemas embebidos
Paralelismo	– SISD (monoprocesadores) – SIMD (matriciales y vectoriales) – MIMD (multiprocesadores y multicomputadores)
Potencia	– Supercomputadores – Servidores de gama alta (Macrocomputadores) – Servidores de gama media – Servidores básicos – Computadores personales (PC) – Computadores móviles

B) Clasificación según su paralelismo

El computador descrito en la Sección 1.2 en un instante dado se encuentra ejecutando una única instrucción con unos datos aislados. Estos sistemas se suele denominar **computadores**

de un único flujo de instrucciones y un único flujo de datos, **SISD** (***Single Instruction stream, Single Data stream***) o computadores von Neumann por seguir los criterios básicos propuestos por este científico en 1946.

Para conseguir mayores velocidades de cómputo se han ideado computadores que siguen el esquema de von Neumann en el sentido de que procesan las instrucciones de una en una, pero cada una de ellas opera con múltiples datos. De esta forma se obtiene una gran eficiencia cuando deben operar con datos vectoriales o matriciales, que se encuentran con mucha frecuencia en aplicaciones científicas y técnicas. Este tipo de sistemas se conocen con el nombre de **Computadores de un único flujo de instrucciones y múltiples flujos de datos**, o **SIMD** (*Single Instruction stream, Multiple Data stream*).

Con objeto de mejorar aún más las prestaciones en la actualidad es frecuente que un mismo computador disponga de diversos procesadores centrales, denominándose **sistema multiprocesador**. Estos computadores suelen utilizar una memoria principal compartida. En los multiprocesadores hay dos alternativas de funcionamiento; en la primera de ellas cada procesador ejecuta un programa o tarea distinta, y en la segunda los distintos procesadores ejecutan en paralelo diferentes módulos en los que se subdivide un mismo programa o tarea (programación y ejecución paralela).

Otra forma de conseguir gran potencia de cálculo es construyendo sistemas compuestos por distintos computadores completos trabajando en paralelo e interconectados adecuadamente entre ellos (**sistemas multicomputador** o ***cluster***). En estos sistemas la memoria principal no es común y los distintos módulos de un programa (o distintos programas de una misma aplicación) intercambian información por medio de mensajes a través de una red de interconexión, por lo que también se conoce como **multicomputadores de paso de mensajes**.

Los multiprocesadores y multicomputadores se dice que son **Computadores de múltiples flujos de instrucciones y múltiples flujos de datos**, o **MIMD** (*Multiple Instruction stream, Multiple Data stream*) ya que en un instante dado se pueden estar ejecutando múltiples instrucciones con múltiples datos.

En el Capítulo 12 se amplían los conceptos básicos sobre paralelismo en computadores.

C) Clasificación atendiendo a la potencia

Una clasificación muy frecuente se hace atendiendo a la potencia o capacidad de cómputo del sistema. Esta clasificación es muy difusa, y se efectúa atendiendo fundamentalmente a parámetros tales como velocidad de funcionamiento, longitud de palabra, capacidad de la memoria principal y número de usuarios que pueden trabajar simultáneamente (Tabla 1.4).

Para comprender bien esta clasificación en primer lugar introduciremos el concepto de servidor. Un **servidor** *es un computador conectado a una red de transmisión de datos que da servicio compartido a múltiples usuarios*. Existen **servidores de aplicaciones**, en los que los usuarios a través de una red pueden ejecutar programas, acceder a la información de bases de datos, acceder a correo electrónico, etc. y servidores que preferentemente ofrecen recursos específicos a compartir a través de la red. Así un **servidor de archivos** es un computador y sistema de almacenamiento dedicado preferentemente a almacenar archivos y donde cualquier usuario de la red puede almacenarlos; un **servidor de impresión** es un computador en la red que gestiona una o varias impresoras, un **servidor de red** es un computador que gestiona el tráfico de información en la red; un **servidor de base de datos** es un computador que procesa la gestión y los accesos a una base de datos; etc.

Tabla 1.4. *Clasificación de los computadores de uso general atendiendo a sus prestaciones (se indican valores aproximados y típicos).*

	Orden de magnitud del precio (dólares)	Nº de procesadores	Capacidad de memoria principal	Orden de magnitud de disco	Nº de usuarios (simultáneos)	Objetivo fundamental
Supercomputador	Más de 5.000.000 $	32 a miles	TB	Centenas de TB	Decenas a miles	Cálculo intensivo de tipo científico y técnico
Servidor de gama alta (microcomputador)	Más de 500.000 $	2 a 128	GB	TB	Cientos a miles	Acceso a grandes bancos de datos desde muchos terminales
Servidor de gama media	25.000 a 500.000 $	1 a 32	Centenas de MB	Centenas de GB	Decenas a cientos	Aplicaciones múltiples en departamentos o empresas de tipo medio a través de red
Servidor básico	Menos de 25.000 $	1 a 8	Centenas de MB	Centenas de GB	Decenas	Aplicaciones múltiples en departamentos o empresas pequeñas a través de red
Computador personal (PC)	500 a 10.000 $	1 a 2	Centenas de MB	Decenas de GB	1 (personal)	Aplicaciones múltiples con un solo usuario
Computador móvil	100 $	1	Varios MB	No tienen	1 (personal)	Asistentes digitales personales (PDA) Computadores de bolsillo Comunicadores personales Calculadoras programables de bolsillo

A continuación describimos distintos grupos de computadores atendiendo a su potencia (de mayor a menor).

Supercomputadores

La característica fundamental de este tipo de sistemas es su rapidez, con varios procesadores o unidades centrales trabajando en paralelo (son sistemas multiprocesador o multicomputador, con de 32 a miles de procesadores), y que pueden ejecutar miles de millones de instrucciones por segundo (**Teraflops**[3]). Su coste va de 10 a 500 millones de dólares o más. Estos computadores son necesarios para **aplicaciones de cálculo intensivo** o **HPC** (*High Performance Computing*), que requieren la realización de un extraordinario número de cálculos a gran velocidad con grandes cantidades de datos de tipo real, necesarios en la resolución de muy diversos problemas de tipo científico y de ingeniería tales como el diseño de nuevas medicinas, simulación del modelo de polución de una gran ciudad (que involucra a cientos de

[3] Según indicamos en la Tabla 1.1: un **Gigaflop** son mil MFlops (1GFlop = 10^3 MFlops); un **Teraflop** son un millón de MFlops (1TFlop = 10^6 MFlops); un **Petaflop** son mil millones de MFlops (1PFlop = 10^9 MFlops); y un **Exaflop** son un billón de MFlops (1TFlop = 10^{12} MFlops).

miles de variables), modelado de fisión nuclear, simulación del enfriamiento de las galaxias, grandes servidores web, etc. Una aplicación típica es la predicción climatológica, ya que ésta requiere efectuar cálculos muy complejos (que simulan matemáticamente el comportamiento de la atmósfera) con grandes cantidades de datos provenientes de multitud de terminales (del orden de 10.000) ubicados en estaciones meteorológicas dispersas a lo largo del mundo, y a tiempo para poder realizar las predicciones; así, para poder realizar predicciones con 72 horas de antelación es necesario un computador de aproximadamente 10 GFlots con 1 Gigabyte de memoria principal, incrementándose notablemente la tasa de ejecución de instrucciones necesaria para predicciones con mayores antelaciones. Para simular el flujo de aire en torno a un ala, para el diseño de un nuevo modelo de avión, con tiempos de ejecución del orden de 15 minutos se requiere un supercomputador (no disponible en la actualidad) con una tasa de ejecución de instrucciones del orden de un billón de MFlops (1 **Exaflops**).

El computador más potente del mundo en Enero del 2006 era el BlueGene/L–eServer construido por IBM e instalado en el Lawrence Livermore National Laboratory de la Universidad de California. Este supercomputador contenía 131.072 procesadores PowerPC 440 de 700 MHz (cada uno de ellos de 2,8 GFlops), adquiriendo una tasa máxima de ejecución de instrucciones de 280 TFlops. Entre las aplicaciones principales de este sistema se encuentra la simulación biomolecular (plegamiento de proteínas, por ejemplo). A partir de la referencia [wTOP500] puede accederse a las características más notables y aplicaciones de los 500 computadores más potentes del mundo[4].

Servidores de gama alta o macrocomputadores (*Mainframes*)

Son grandes computadores de uso general con amplias posibilidades de procesamiento, gran memoria y capaces de admitir desde varios cientos a miles de usuarios trabajando simultáneamente. Son utilizados por instituciones que procesan la información de grandes bases de datos (grandes servidores web, servicios de proceso de datos de entidades bancarias, reservas de billetes de compañías aéreas, etc.). Otras características típicas de estos computadores es procesar la información en modo de **transacciones** (véase Sección 13.5.3), con gran cantidad de usuarios conectados a través de redes y disponer de una gran capacidad de memoria masiva (del orden de Terabytes). Su precio se encuentra en el rango de 1 a 10 millones de dólares y suelen ser multiprocesadores o muticomputadores de 2 a 128 procesadores.

Servidores de gama media y básica

Son equipos para utilizar interactivamente por múltiples usuarios simultáneamente, similares a los macrocomputadores, pero a escala reducida de prestaciones y precio, y suelen ser utilizados en empresas o departamentos de tipo medio o pequeño.

Se configuran como monoprocesadores o multiprocesadores que pueden llegar a tener del orden de GB de memoria principal, cientos de GB de disco, y actúan interconectados en una red de área local o de gran área (Internet), pudiendo atender simultáneamente decenas de accesos de estaciones de trabajo, PC o terminales conectados a la red.

Los servidores de gama media pueden llegar a tener hasta 32 procesadores y su coste oscila, aproximadamente, entre 50.000 y 1.000.000 de dólares. Los de gama baja o servidores básicos, por su parte, no suelen contener más de 8 procesadores, estando comprendido su precio entre 1.000 y 100.000 dólares.

[4] En enero de 2006, el octavo supercomputador más potente del mundo era el *Mare Nostrum*, de 28 TFlops e instalado en el Centro de Supercomputación de Barcelona (España).

Computadores Personales (PC, *Personal Computers*) (véase Sección 12.6)
Son sistemas monousuario, usualmente con centenas de MB de memoria principal, y equipados con disco magnético con decenas de GB, unidad CD-ROM, MODEM, tarjeta de sonido, teclado, pantalla y otros periféricos. Es el tipo de computadores de uso general más difundido y entre sus características se encuentran la gran cantidad de programas disponibles para ellos, y la gran compatibilidad entre unos y otros. Suelen utilizar el sistema operativo Windows-MS, Linux o Mac OS. Los de gama superior se conocen como **estaciones de trabajo** (*Workstations*); inicialmente fueron proyectadas para aplicaciones que requieren mucho cálculo y una alta capacidad gráfica, necesitando por tanto procesadores potentes y pantallas de alta calidad, como pueden ser las de diseño con ayuda de computador, las de animación, etc. (véase Sección 1.8).

Los computadores incluidos en este grupo suelen montarse bajo las siguientes formas:

- **Torre***:* la altura del chasis es mayor que sus otras aristas y el equipo se puede ubicar en el suelo
- **Sobremesa***:* la altura del chasis es la menor de sus aristas, y por lo general se ubica encima de una mesa
- **Portátil (*notebook*):** el computador está miniaturizado (aproximadamente ocupa una superficie de 22 por 29 centímetros) y diseñado de forma que su peso (del orden de 2 Kg.) y consumo sean muy reducidos.

Los computadores personales suelen ser monoprocesadores, aunque los más potentes son biprocesadores, y su precio oscila entre los 600 y 1.500 euros, siendo los más caros los portátiles.

Computadores móviles
Se caracterizan por su pequeño tamaño (aproximadamente el de una agenda, 20x10x4cm), peso reducido (unos 300g) y alimentación por acumuladores (pilas); todo ello para obtener una gran movilidad. En este grupo incluimos, además de las calculadoras programables de bolsillo, una serie de sistemas que en la actualidad están proliferando notablemente y que se suelen utilizar para aplicaciones tales como creación de pequeñas hojas de cálculo, agenda de direcciones, planificación horaria, directorio telefónico. No disponen de unidades convencionales de disco, usando **memorias flash** y algunos disponen de **discos de estado sólido** *(**SSD**, Solid State Disks*) que son del tamaño de una tarjeta de crédito, y de muy baja capacidad; sin embargo suelen poder conectarse a computadores más potentes o a Internet, para intercambio de datos. Entre otros sistemas aquí podemos incluir:

- **Asistentes digitales personales (Personal Digital Assistant, PDA).** Suelen estar provistos de un lápiz electrónico para escribir sobre una pantalla sensible al tacto (Sección 11.2.2.1), y contienen programas de reconocimiento de caracteres. Frecuentemente utilizan el sistema operativo Windows CE.
- **Computadores de bolsillo (Palmtop Computers), organizadores o agendas personales (Personal Organizers).** Son similares a los PDA pero suelen disponer de mayor RAM, en lugar de la pantalla sensible al tacto tienen un pequeño teclado para introducir tanto datos numéricos como texto, siendo el campo de sus aplicaciones más amplio que el de los PDA.
- **Comunicadores personales.** Son sistemas proyectados para tener acceso a Internet. Entre ellos podemos incluir los **teléfonos WAP** *(Wireless Application Protocol*, o **protocolo para comunicaciones inalámbricas**) que permiten enviar fax, acceso a Internet para uso de correo electrónico y consulta de páginas web, por ejemplo.

- **Calculadoras programables de bolsillo.** Su entrada es un teclado sencillo que incluye teclas para introducción de datos sólo numéricos y teclas específicas para cada instrucción, su salida esta constituida por un visualizador óptico ("display"), y dispone de una capacidad de memoria y lenguaje de programación muy limitados. Fabricantes de estos equipos son Casio y Hewlett Packard.

Los tipos de computadores citados anteriormente (desde supercomputadores a computadores móviles) corresponden a los computadores de uso general; por lo que a ellos habría que añadir los computadores de uso específico, fundamentalmente computadores embebidos construidos con microprocesadores o microcontroladores y ya citados al inicio de esta sección.

Hay que hacer notar que la clasificación realizada tiene límites muy difusos. Con los avances tecnológicos y desarrollo de nuevas arquitecturas, un computador considerado de pequeña capacidad en la actualidad tiene mayor capacidad y es más veloz que una considerada grande hace tan sólo unos diez años (véase Capítulo 20). Además la diferenciación es con el paso del tiempo más cuantitativa que cualitativa en el sentido de que prácticamente la totalidad de los sistemas se construyen con microprocesadores y utilizando los mismos conceptos de diseño, obteniéndose las diferencias de rendimiento por la utilización de componentes de menor o mayor velocidad y capacidad de almacenamiento, con menor o mayor grado de paralelismo.

1.6 Niveles conceptuales de descripción de un computador

El **hardware** o **soporte físico** de un computador es el conjunto de los componentes que integran su parte material; es decir, el conjunto de circuitos electrónicos, cables, armarios, dispositivos electromecánicos y otros elementos físicos que conforman el computador.

El **software** o **soporte lógico** de un computador es el conjunto de programas (del sistema operativo, de utilidades y de los usuarios) ejecutables por el computador.

Para que un computador funcione es necesario utilizar programas; es decir, un computador con tan sólo sus elementos físicos no funciona: tan imprescindible es el hardware como el software.

La palabra hardware no sólo se utiliza para designar los dispositivos físicos del computador (tecnología) y su interconexión (arquitectura), sino también todo lo relacionado con ellos. Así, por ejemplo, la Teoría de la Conmutación, la Electrónica, etc, son disciplinas relacionadas directamente con el hardware. Lo mismo puede decirse del software. Software no son sólo los programas de un computador concreto, sino que también se considera software todas las materias relacionadas con la construcción de los programas: organización y estructuración de los datos, construcción de algoritmos, análisis de aplicaciones, metodología utilizada para redactar programas, etc.

El computador es uno de los sistemas más complejos ideados por el hombre; en efecto, contiene cientos o miles de circuitos integrados pudiendo contener cada uno de ellos millones de elementos individuales (transistores) todos ellos actuando coordinadamente. Para facilitar su diseño o análisis, se puede describir según distintos niveles conceptuales. La distinción entre niveles más sencilla es la que hay entre software y hardware. Una distinción más detallada (de nivel de mayor complejidad al de menor complejidad) es la que se muestra en la Figura 1.8.

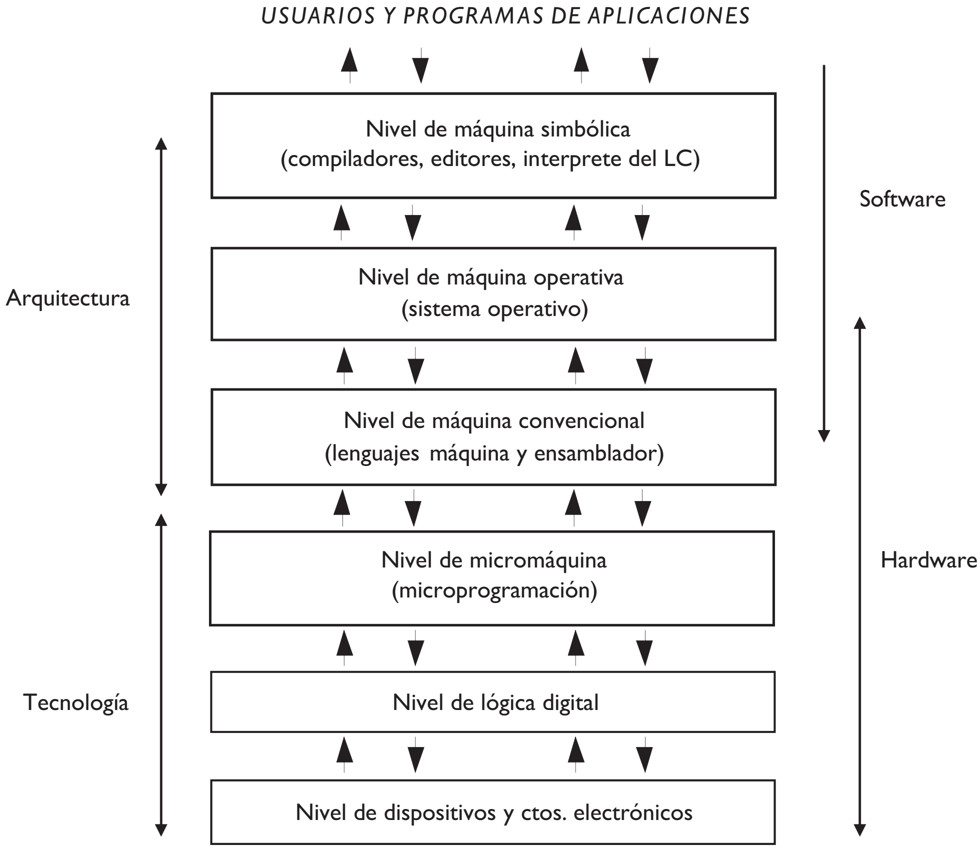


Figura 1.8. *Niveles conceptuales de descripción de un computador.*

El estudio o diseño dentro de cada nivel se efectúa utilizando o *viendo* al computador según las primitivas proporcionadas por el nivel inmediato inferior. Así, para diseñar el sistema operativo se considera al computador en el nivel de máquina convencional; es decir, en teoría no sería necesario conocer niveles inferiores a éste (el de lógica digital, por ejemplo).

Los niveles 1, 2 y 3 corresponden a los de **tecnología de computadores**, y son de una gran importancia ya que el crecimiento del rendimiento de los computadores en gran medida depende de ella. Por ejemplo, en las décadas de los 70 y los 80 se obtuvo un incremento del 18 al 35 % por año en el rendimiento de los computadores gracias a mejoras tecnológicas. Los elementos del nivel electrónico son componentes tales como transistores, condensadores y resistencias; y la información se representa con valores de tensiones eléctricas (0/3.5 V, por ejemplo), corriente eléctricas (0, 16 mA) o estados de magnetización (Norte, Sur), etc. La mayoría de las operaciones básicas de la máquina se describen a este nivel y sus elementos básicos (puertas lógicas) pueden almacenar, manipular y transmitir datos representables en forma binaria (0, 1). Los capítulos 5 y 6 se dedican a sentar las bases tecnológicas de los computadores que posibilitan la descripción de los mismos en los niveles 1 y 2.

En el Capítulo 7 se describirán algunos tópicos de la estructura de computadores dentro del Nivel 3 (microoperaciones). Los módulos que utiliza este nivel son registros, multiplexores, bancos de registros, módulos de memoria RAM, etc. construidos con los módulos del Nivel 2

(puertas lógicas). Las señales son para realizar operaciones tales como carga de un registro, efectuar una operación concreta en la ALU, escribir en la memoria principal, etc. El Capítulo 9 se dedicará a profundizar en la descripción de los computadores en este Nivel 3.

El Nivel 4, lenguaje máquina, es el genuino nivel de **arquitectura del computador** ya que el diseño de un computador parte de la especificación de las instrucciones del lenguaje máquina. A este nivel se dedicará el Capítulo 8. El lenguaje ensamblador es igual al lenguaje máquina, salvo que utiliza una terminología distinta para describir los módulos o primitivas (las instrucciones) que se definen en este Nivel 4.

El Nivel 5, sistema operativo, puede considerarse como la interfaz entre el hardware y software. Añade una capa para facilitar el uso del hardware y hacerlo lo más eficaz posible desde el punto de vista de los usuarios y de los programas de aplicación. A él se dedica el Capítulo 13.

El último nivel (Nivel 6, o de máquina simbólica) es el que realmente ven los programadores de aplicaciones y los usuarios, y está formado básicamente por los traductores o, en general, procesadores de lenguajes. Si está bien concebido, el usuario no tendrá necesidad de conocer nada de los niveles inferiores, y sus primitivas y lenguaje de utilización son simbólicos, debiendo ser lo más próximos posibles al hombre (Capítulo 14).

1.7 Utilización de los computadores

En esta sección se dan unas ideas sobre la sistemática o metodología que usualmente se sigue para la realización de una aplicación con computador. También se introducirá la terminología básica relacionada con la estructura o planificación de los datos en informática. Estos temas se amplían en capítulos posteriores (Capítulos 3 y del 13 al 17).

1.7.1 Metodología

En la Sección 1.1 se comentó que el usuario indica al computador qué es lo que tiene que hacer por medio de programas. Es decir, para usar un computador se deben expresar todas las operaciones o tratamientos a efectuar *desmenuzados* en instrucciones del repertorio del lenguaje de programación que se va a utilizar.

Para resolver un problema (confeccionar la nómina de una empresa, por ejemplo) con computador es necesario definir y describir minuciosamente con rigor cómo debe resolverse el problema. Para ello se suelen seguir las siguientes fases:

Planteamiento o definición del problema
Consiste en describir claramente qué es lo que se pretende, incluyendo la presentación (*formato*) de los resultados que se desean obtener. Por ejemplo, en el caso de la nómina, el proceso puede consistir en generar los siguientes documentos:

- Relación de empleados en la que se especifique el banco, cuenta corriente y cantidad que debe percibir cada uno de ellos.
- Listado en que figure la cantidad retenida, por impuestos oficiales, a cada empleado.
- Hoja para cada empleado, detallando su liquidación.
- Listado sobre las cuotas a la Seguridad Social, etc.

En función de la documentación a obtener, habrá que determinar qué datos debe utilizar el computador para hacer los cálculos intermedios oportunos y producir los listados. Esos datos

podrían ser: nombres y apellidos de cada empleado, categoría laboral (en función de la cual se obtendrá el sueldo), antigüedad, complementos retributivos, número de identificación del empleado, número de la seguridad social, entidad bancaria y número de cuenta corriente, etc.

En el planteamiento del problema interviene el usuario final del programa además de especialistas en informática. En el *planteamiento* (o definición de necesidades) deben especificarse las reglas o pasos que es necesario seguir para resolver el problema general o los aspectos parciales, como puede ser el método para obtener lo que cada empleado debe abonar a la seguridad social. Se trata de definir los algoritmos inherentes a la aplicación. Un **algoritmo** *es un conjunto ordenado de reglas o instrucciones tal que siguiéndolas paso a paso se obtiene la respuesta a un problema dado, sean cuales sean los datos o circunstancias particulares del mismo*.

En esta fase se especifica también la *interfaz de usuario* o forma en que se va a utilizar la aplicación: con menús de opciones, etc.

Análisis

La responsabilidad de esta fase es de los especialistas en informática (*analistas de aplicaciones*), que trabajan con los usuarios finales de la aplicación. El **análisis** *consiste en el estudio por un procedimiento lógico de la solución del problema, descomponiéndolo en diferentes fases realizables con el computador y los equipos disponibles*. El análisis, partiendo del planteamiento, determina cómo deben estructurarse los datos de entrada, cómo deben depurarse los errores de dichos datos, qué archivos de datos deben definirse, qué lenguaje de programación es adecuado utilizar, si se debe emplear alguno de los programas de utilidad existentes en la instalación, etc.

El analista descompone el problema global en problemas más simples e interrelacionados, ya que los resultados de unos pueden ser los datos de entrada de otros. Cada uno de estos problemas se podrá resolver con un programa independiente, de esta forma el análisis consigue descomponer el problema inicial en módulos sencillos.

El analista produce un esquema, que se suele describir en un lenguaje algorítmico o representar en una gráfica denominada ***organigrama, diagrama de flujo,*** u ***ordinograma***. Un organigrama se construye con unos símbolos normalizados, con los que se muestra una panorámica de la aplicación en función de la naturaleza de los datos de entrada y de los archivos de datos, de los módulos o programas a realizar y sus interfaces, de la información de salida a generar y de dónde se quiere ubicarla (en disco, pantalla o impresora; por ejemplo).

Programación

Consiste en describir los módulos o programas, definidos en el análisis, por medio de instrucciones del lenguaje de programación que se utiliza (instrucciones de E/S, transferencias de información, cálculos aritmético-lógicos y bifurcaciones condicionales, etc.). Esta descripción se puede efectuar con ayuda de un organigrama o utilizando una terminología específica de programación denominada **pseudocódigo**. Posteriormente el *programador* redacta el programa, codificando las instrucciones según el repertorio y reglas del lenguaje de programación.

Prueba y depuración de la aplicación

Esta etapa consiste en comprobar el funcionamiento de cada programa individual con datos ficticios y reales. Una vez depurados los programas individuales, se prueba por separado cada fase de la aplicación, y, en su caso, la aplicación completa. A veces la etapa de experimentación obliga a efectuar una revisión de las etapas anteriores, incluso la de planteamiento, con objeto de obtener unos resultados satisfactorios. En general cuando en una cualquiera de las

etapas no se obtienen resultados satisfactorios suele volverse a la etapa o etapas anteriores para revisarlas.

Explotación de la aplicación
En esta fase los programas se utilizan, produciéndose la información requerida (en el caso de la nómina, mensualmente).

Las etapas descritas corresponden a la metodología seguida en la mayoría de las aplicaciones realizadas con computador. Hay situaciones en que esta sistemática no es válida o no es útil. Así, en aplicaciones de investigación científica, frecuentemente el usuario final (el investigador) debe encargarse del análisis, de la programación y de la ejecución de la aplicación. Además, en este caso, los programas a veces son para utilizarlos una o muy pocas veces (cosa que no ocurre en aplicaciones de gestión), ya que se trata, por ejemplo, de resolver una ecuación matemática concreta, verificar una determinada hipótesis o comprobar una conclusión teórica. Una vez obtenido el resultado, no tiene sentido repetir frecuentemente o periódicamente (como sucede en el caso de la nómina) la ejecución del programa.

Existen **bibliotecas de programas** y paquetes de **programas de utilidad** (denominados genéricamente **utilidades**), que se adquieren con el computador o en empresas especializadas en producción o venta de software, y que se incluyen en el software del computador. Estos paquetes son de naturaleza muy diversa: matemática, estadística, de gestión, etc., y son de una gran ayuda para la utilización del computador, evitando o reduciendo el trabajo de análisis y programación.

Hasta la década de los setenta la mayoría de las aplicaciones informáticas eran desarrolladas *a medida* para empresas o usuarios concretos. En la mayoría de los casos la propia empresa disponía, en su *servicio de informática*, de una plantilla específica de profesionales informáticos (analistas, programadores, operadores de computador y operadores de equipos de entrada de datos). Posteriormente se desarrollaron programas o paquetes software (*productos de programación*) para ser usados por numerosas empresas y personas, y en cuya realización interviene también gran cantidad de especialistas (proyectistas, analistas, programadores, etc.). Así, los paquetes software pasan a ser auténticos productos comerciales, desarrollados por gran número de personas y distribuidos a gran escala.

Esto obligó a completar la metodología de realización de aplicaciones vista anteriormente, con el nuevo enfoque del desarrollo y mantenimiento de programas considerados como productos tecnológicos. Surge así (a finales de los 60) el concepto de ingeniería del software, a la que se dedicará el Capítulo 17. En definitiva se trata de aplicar las técnicas habituales de la ingeniería a la producción del software, con objeto de optimizar los costes, facilitar el mantenimiento, aumentar la comodidad de uso, y mejorar, en definitiva, la calidad de los *productos de programación*.

La **Ingeniería del Software** puede definirse como la *disciplina que trata de los aspectos tecnológicos y burocráticos relacionados con el diseño, producción y mantenimiento sistemáticos de programas de computadores.*

Los aspectos burocráticos citados incluyen facetas tales como estimación de costos, asignación de personas a la realización de un proyecto, forma de realizar la documentación, mentalización de los usuarios para la aceptación del producto, etc.

Desde el punto de vista de la ingeniería del software el *ciclo de vida de un producto* tiene tres grandes fases: *definición, desarrollo y mantenimiento*; que pueden descomponerse en las siguientes cinco:

1. **Análisis**, donde se incluyen dos aspectos: a) el planteamiento del proyecto y la estimación de costos, y b) la definición de requisitos.
2. **Diseño**, consiste en la concepción y planificación del producto.
3. **Implantación**, comprende la plasmación de las especificaciones de diseño en un programa de computador (en un lenguaje de alto nivel).
4. **Prueba**, que implica normalmente dos aspectos:
   a) la *verificación* o comprobación de si se está realizando correctamente lo que se pretendía (equivale al control de calidad), y
   b) la *validación* o comprobación de si se está o no realizando el producto adecuado.
5. **Mantenimiento**, se produce en la etapa en que el producto se está utilizando por el usuario, para obtener de él el provecho para el que fue concebido. Los objetivos del mantenimiento son:
   a) corregir defectos o errores,
   b) readaptar el producto a nuevas especificaciones o circunstancias, y
   c) mejorar el producto (más velocidad, menos requerimientos de recursos hardware, mayor comodidad de uso, nuevas prestaciones, etc.).

A veces antes de hacer el producto definitivo se realizan las cuatro primeras fases con un ***prototipo*** que se presenta al usuario para su validación. Una vez realizada ésta se repiten las fases citadas para el desarrollo del producto definitivo.

El Capítulo 17 de este libro se dedicará a ampliar los conceptos expuestos en esta sección.

1.7.2 Organización de los datos

Los datos e informaciones que se dan o almacenan en un computador, no se introducen y guardan sin orden y al azar, sino que se estructuran y planifican de una forma adecuada, según un determinado formato. Esta organización es decidida por los analistas y programadores, y debe ser tenida en cuenta en las instrucciones de lectura y escritura de los programas que utilizan o generan dicha información.

Los datos se pueden organizar en el computador de acuerdo con distintas estructuras, que se estudiarán en el Capítulo 2. Algunas de estas estructuras son: arrays, tablas, árboles, listas y anillos.

Aquí únicamente se van a incluir unos conceptos y definiciones básicos, relacionados con el almacenamiento de datos en la memoria masiva, que serán convenientemente ampliados en el Capítulo 15.

Archivo o **fichero** es un conjunto de información del mismo tipo (homogénea) referente a unos determinados elementos, tratada como una unidad de almacenamiento y organizada de forma estructurada para la recuperación de un elemento o dato individual.

Ejemplos de archivos son:

- Archivo de historias clínicas de enfermos de un hospital determinado.
- Archivo de empleados de una empresa.
- Archivo de alumnos de un centro docente.
- Archivo de datos metereológicos diarios de una localidad.
- Archivo de libros de una biblioteca.

En la Figura 1.9 se tiene parte del contenido de un archivo relativo al control de impuestos y multas de tráfico.

Matrícula coche	Marca	Modelo	Propietario	Domicilio	Pagó impuesto municipal	Nº de multas	TOTAL DEBE
BCD1725	Ford	Megane	Juan Llor Isla	Ancha Nueva, 16	No	0	25.500
CBG1823	Volvo	720	Pedro Sosa Ca	Recogidas, 23	Sí	5	38.400
DCR1732	Fiat	Croma	José Pi Ros	Acapulco, 13	No	3	0
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
CBD7784	Peugeot	607	Marta Prieto	Recogidas, 16	Sí	0	12.324

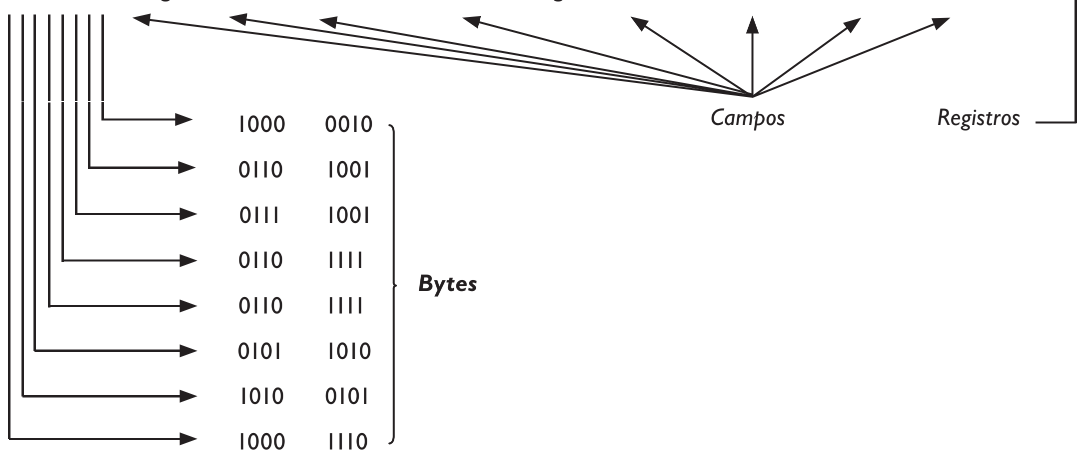


Figura 1.9. *Ejemplo de archivo para control de multas de tráfico.*

Por lo general los datos que lee o genera un programa se estructuran en forma de archivo de datos. El archivo de datos de entrada (el que lee el programa) puede ser introducido en la memoria masiva con anterioridad a la ejecución del programa. El sistema operativo considera a los archivos como unidades de almacenamiento incluyendo órdenes tales como *crear un archivo, copiar un archivo* y *borrar un archivo.* Además existen programas específicos para el mantenimiento o puesta al día de archivos para que siempre estén actualizados. Estos últimos programas, denominados **editores** o **gestores de archivos**, insertan, eliminan o modifican datos del mismo.

Los **registros** son las *estructuras o unidades que forman el archivo y que contienen la información correspondiente a cada elemento individual.* Así, en los ejemplos anteriores los registros son:

- La información contenida en una historia clínica.
- Los datos referentes a un empleado determinado.
- Los datos de un determinado alumno.
- Los datos metereológicos de un día.
- Los datos referentes a un libro.

En la Figura 1.9, cada línea corresponde a un registro, ya que incluye la información relativa a un vehículo. El registro corresponde a la información contenida en una ficha de un archivo o archivo manual.

Campo, artículo o **ítem** *es una variable o dato que forma parte de un registro y representa una información unitaria o independiente*. Son campos, por ejemplo:

- El nombre de un enfermo. La edad del enfermo. Su lugar de nacimiento. El domicilio. El código de la cama donde ha estado (o está) hospitalizado. Cada uno de los síntomas detectados en él, etc.
- El nombre del empleado. Su número de identificación. Fecha de alta de la empresa. Puesto que ocupa. Etc.
- La temperatura máxima. El código de identificación de la estación metereológica. La fecha. La temperatura mínima. Etc.
- El autor o autores del libro. La editorial. El código de materia. Cada palabra clave de identificación de su contenido. El número de páginas. El lugar donde se encuentra. Etc.

En la Figura 1.9 cada columna es un campo (matrícula, marca, modelo, propietario, domicilio, pago impuesto municipal, número de multas y cantidad adeudada).

Los campos se componen de caracteres (Sección 1.1), que son los símbolos más elementales utilizados para definir el contenido de un campo. Recordemos que el carácter se codificaba en el interior del computador ocupando un byte (Figura 1.9).

Una misma empresa o centro puede efectuar diversas aplicaciones que utilizan los mismos datos. La sección de personal es posible que tenga en el computador un archivo de empleados conteniendo el número de identificación del empleado, nombre, dirección, puestos que ha ocupado y ocupa, experiencias previas, etc. La sección de contabilidad puede mantener en el computador otro archivo sobre empleados conteniendo el número de identificación, nombre, domicilio, puesto que ocupa, complementos que debe percibir en nómina, etc., de cada uno de ellos. Estos archivos están organizados y estructurados según la aplicación que los utilizan; por ejemplo, el de la sección de contabilidad se proyecta de forma que contenga los datos necesarios para producir la nómina. Una primera objeción al planteamiento anterior es que hay que introducir mucha información común en ambos archivos, que, en el ejemplo anterior estará duplicada en la memoria del computador (el número de identidad, el nombre del empleado y el domicilio, por ejemplo, figuran en ambos archivos). Estas duplicidades dan lugar, si no se evitan, a que se tarde más o se ocupen más personas en introducir los datos; sea más complejo depurar de errores los datos; se ocupe más memoria masiva; haya que mantener actualizados simultáneamente dos archivos, etc.

Es más lógico introducir los datos de una sola vez (evitando redundancias innecesarias), diseñando una estructura según la organización global de la empresa y sus necesidades actuales y futuras. Esta estructura, o **base de datos**, no tiene por qué depender (como en el ejemplo) de una aplicación o programa concreto, sino que se diseña considerando la globalidad de posibles aplicaciones.

Más formalmente, una **base de datos** *es una colección de archivos que están interconectados lógicamente* o, más rigurosamente, *es una organización sistemática de datos orientada a facilitar su acceso, recuperación y actualización, por diversos programas o usuarios, y que incluye las relaciones de unos datos con otros.* Las bases de datos se crean, actualizan y utilizan por medio de programas especiales denominados **gestores de bases de datos**. El software gestor de la base de datos que facilita su creación, mantenimiento y uso se

denomina **sistema de gestión de la base de datos** (**DBMS**, “Database Management System”), y está constituido por dos partes; una que permite al programador diseñar e implementar las actividades de mantenimiento (inserción, borrado y modificación de los registros que constituyen la base de datos) y otra que permite recuperar información contenida en la base de datos. Estas operaciones se efectúan utilizando tres lenguajes: **lenguaje de definición de datos, lenguaje de manipulación de datos** y **lenguaje de consulta** (**SQL**, *Structured Query Language*). Este último lenguaje permite utilizar eficientemente la base de datos a personal no especializado en informática. Ejemplos de sistemas de gestión de base de datos son el Microsoft Access y los desarrollados por la empresa Oracle.

En los Capítulos 15 y 16 se ampliarán los conceptos sobre archivos y bases de datos que acabamos de ver.

1.8 Aplicaciones de la informática

Es bien conocido que uno de los agentes más importantes de la sociedad actual es la información; de ahí el gran desarrollo e interés de la informática, que tiene por objeto *el tratamiento automático de la información* (Sección 1.1). Hay pocas actividades humanas en que no tenga incidencia, de forma directa o indirecta, la informática.

Los computadores resultan útiles para aplicaciones que reúnen una o varias de las siguientes características:

1. Necesidad de un *gran volumen de datos*. Los computadores resultan particularmente adecuados para procesar grandes cantidades de datos (por ejemplo, datos relacionados con los clientes de una entidad bancaria).
2. *Datos comunes*. Las bases de datos (Sección 1.7.2 y Capítulo 16) posibilitan que los datos incluidos en un computador puedan utilizarse en múltiples aplicaciones, sin necesidad de que estén físicamente repetidos. Ello, además de ahorrar tiempo en la introducción de los datos y ahorrar espacio en la memoria masiva, facilita considerablemente la actualización de los mismos. Por ejemplo, una vez introducida la información sobre un profesor de una universidad, esta podría ser utilizada, sin necesidad de replicarla, para distintas aplicaciones, tales como abono de nómina, realización de horarios de clase, control de carga docente del profesorado, elaboración de censo para elección de órganos de gobierno, etc.
3. *Repetitividad*. Tal vez una de las características más relevante de los computadores (y sus lenguajes de programación) es procesar ciclos de instrucciones iterativamente. Una vez programada las instrucciones que constituyen un ciclo y el número de iteraciones, el computador las ejecutará sin importar lo elevado de dicho número. También un mismo programa o rutina puede ser ejecutado con gran cantidad de datos (piénsese, por ejemplo, para calcular los intereses de las cuentas corrientes de los clientes de un banco, un programa de cálculo de intereses relativamente sencillo se aplica a una gran cantidad de datos).
4. *Distribución*. El origen y destino de la información no necesita estar ubicado en el computador central. En efecto, la información que procesa un computador puede introducirse (u obtenerse los resultados) a través de terminales distribuidos por áreas geográficas muy extensas (desde distintas habitaciones del edificio de una empresa hasta estaciones repartidas por todo el mundo conectados a través de líneas telefónicas). También puede procesarse la información en distintos computadores distribuidos en red.

5. Un computador puede realizar todas sus operaciones con una *precisión* controlada, obteniendo resultados consistentes con la precisión de los datos introducidos.
6. *Cálculos complejos*. Utilizando lenguajes de programación adecuados y rutinas de bibliotecas matemáticas, es posible efectuar cálculos sofisticados (resolución de modelos matemáticos atmosféricos para predicción del tiempo, por ejemplo).
7. Las operaciones que realiza un computador las efectúa a una *gran velocidad*, en comparación con los humanos.

Aquellas actividades que requieran o presenten alguna de las características anteriores, son candidatas a ser efectuadas con ayuda de computador.

A continuación se relacionan algunos ejemplos de aplicación de la informática. Estos ejemplos los hemos agrupado en siete apartados, que están basados en la taxonomía de Aplicaciones de los Computadores incluida en las "*CR Categories*" dadas por la *Association of Computing Machinery* (ACM) [ACM].

1. Procesamiento de datos administrativos

Este es el campo de aplicación de los computadores más extendido, y trata de automatizar las funciones de gestión típicas de una empresa, incluyendo aplicaciones o programas para realizar procesos tales como:

- Contabilidad. Control de caja.
- Procesamiento de pedidos. Facturación.
- Control de proveedores y clientes.
- Control de almacén.
- Control de producción y de productividad.
- Gestión de personal. Nóminas.
- Planificación y control de proyectos grandes y complejos.
- Programación lineal: búsqueda de soluciones óptimas; por ejemplo, minimización de costes de transporte,
- Investigación y prospección de mercado.
- Modelos financieros y para predicción (de bolsa, por ejemplo).
- Gestión bancaria (control de clientes, gestión de cajeros automáticos, etc.).
- Sistemas de gestión de ***terminales punto de ventas*** (cajas de abono en almacenes comerciales).
- Gestión bibliotecaria. Archivos automatizados de bibliotecas. Control de préstamos, localización de libros, etc. (véase también "Documentación científica y técnica" en el apartado "Ciencias sociales y del comportamiento").
- Seguros (evaluación de riesgos, etc.).
- Sistema de reserva y expedición de billetes (compañías de transporte aéreas, ferrocarriles, etc.).

En este apartado se pueden considerar los paquetes integrados de **oficina electrónica** o de **ofimática**, que suelen estar compuestos por los siguientes programas:

- Procesador de textos.
- Hoja electrónica.
- Gestión de archivos o/y bases de datos.
- Correo electrónico.

- Agenda electrónica.
- Aplicaciones gráficas.

2. **Ciencias físicas e ingeniería**

   Los primeros computadores se desarrollaron precisamente para aplicarlos en este campo. Se utiliza el computador como instrumento para la resolución de modelos y cálculos matemáticos, tales como:

   - Resolución de ecuaciones y de problemas matemáticos, en general (Cálculo numérico o simbólico).
   - Análisis de datos experimentales utilizando técnicas estadísticas.
   - Simulación y evaluación de modelos (por ejemplo, los utilizados en predicción metereológica).
   - Realización de tablas matemáticas.

3. **Ciencias de la vida y médicas**

   Aquí se incluyen aplicaciones tales como:

   - Investigación médica, biológica y farmacéutica (base de datos sobre el DNA, análisis de datos de experimentos sobre efectos de medicamentos, etc.).
   - Ayuda al diagnóstico (sistemas expertos para diagnóstico médico, por ejemplo).
   - Bases de datos con historias clínicas.
   - Medicina preventiva (control de vacunación de niños, por ejemplo).
   - Electromedicina (véase apartado de "Ingeniería con ayuda de computador").

4. **Ciencias sociales y del comportamiento**

   Algunos ejemplos de aplicaciones en este apartado son:

   - Análisis de datos (evaluación de encuestas, por ejemplo).
   - Bases de datos jurídicas (incluyendo legislación, jurisprudencia, etc.).
   - Aplicaciones en educación:
     - Enseñanza con ayuda de computador (**CAI** –siglas de "Computer Assisted Instruction"– o **CAL** –"Computer Aided Learning"). Consiste en la utilización del computador como herramienta pedagógica, aplicable en gran cantidad de disciplinas.
     - Evaluación automática de exámenes.
   - Juegos con computador (videojuegos, ajedrez, etc.).
   - Documentación científica y técnica: bases de datos con referencias de publicaciones, artículos en revistas, patentes y comunicaciones a congresos, a las que se puede acceder según diversos criterios (materia, autor, año, etc.). A esta información se puede acceder a través de redes de computadores de área local o de internet.

5. **Arte y humanidades**

   Aquí se considera el uso del computador en aspectos tales como:

   - Composición de cuadros.
   - Composición musical.
   - Elaboración de publicaciones tales como libros, periódicos y revistas.
   - Realización de escenas animadas para películas de cine, televisión, etc.
   - Análisis automático de textos (determinación de frecuencias de uso de palabras, etc.).

6. **Ingeniería con ayuda de computador**

   Comprende aplicaciones en las que se usa el computador como herramienta para facilitar diseños de ingeniería, diseño de productos comerciales, trazados de planos, etc. También incluimos aquí el apartado de Informática industrial.

   - **Diseño, fabricación y test con ayuda de computador**. Este campo se suele conocer con las siglas **CAD** (*Computer Aided Design*), ***CAM*** (*Computer Aided Manufacturing*) y **CADMAT** (*Computer Aided Design Manufacturing and Testing*). Usualmente estas aplicaciones se realizan con estaciones de trabajo con altas prestaciones gráficas y permiten la realización automática de tareas tales como análisis y diseño de circuitos integrados, cálculo de estructuras (de edificios y obras públicas, en general), trazado de carreteras, corte de tejidos para confección de prendas de vestir, diseño de coches, etc.
   - Cartografía.
   - Minería (estimación de reservas de un yacimiento a partir de la información obtenida con sondeos, realización de proyectos de extracción, por ejemplo).
   - **Informática industrial**, comprende el uso de computadores para controlar o monitorizar unidades individuales (por ejemplo, el encendido o apagado de una lámpara o de una resistencia de caldeo), máquinas individuales, grandes sistemas de ingeniería e incluso grandes complejos industriales. En este apartado también se encuentran sistemas programables de instrumentación y medida, en general. Hay desde pequeños sistemas embebidos (control del funcionamiento de una lavadora, o de un aparato de aire acondicionado programable) hasta sistemas complejos de industrias de fabricación (control integrado de una petroquímica). Este tipo de aplicaciones se ha visto muy impulsado por el desarrollo de los microprocesadores y microcontroladores, ya que éstos han propiciado que la mayor parte de los sistemas electrónicos actuales sean digitales y sus estructuras se corresponden a las de un computador de uso específico (sistema embebido, Sección 1.5). Algunas áreas incluidas en este campo de aplicaciones son:
     - Instrumentación electrónica.
     - Electromedicina (tomografía axial computerizada, control de una unidad de cuidados intensivos, etc.).
     - Robots.
     - Máquinas herramientas con control numérico.
     - Control de procesos en plantas industriales.
     - Sistema de control computerizado de vehículos (de un automóvil, avión, etc.).
     - Aplicaciones militares (control de dirección de un misil).
     - Control de tráfico, de iluminación, de polución industrial, de redes de distribución de agua, etc.

7. **Computadores en otros campos o sistemas**

   Se consideran aquí otros campos de aplicación no incluidos en los apartados anteriores, y que son de interés general, como el uso de un computador como sistema de composición de textos (procesador de textos) o como sistema de comunicación (correo electrónico, fax, etc.).

La lista de aplicaciones dada anteriormente sólo pretende hacer recordar al lector el amplio abanico de posibilidades de la informática. Además se han desarrollado nuevas metodologías

y técnicas que han ampliado notablemente el abanico de posibles aplicaciones de la informática. A continuación se comentan, a título de ejemplo, los conceptos de inteligencia artificial, informática gráfica, aplicaciones multimedia e internet.

La **inteligencia artificial** puede considerarse como una metodología ideada para el uso de computadores para simular varios aspectos del comportamiento inteligente. Estos aspectos incluyen el razonamiento deductivo, la visión, la comprensión del lenguaje natural, el habla, la resolución de problemas, el modelado del comportamiento humano. Puede decirse que los programas de Inteligencia Artificial responden ante una situación o datos, como previsiblemente lo haría una persona inteligente. Tal vez el modelo más genuino creado dentro de la disciplina de la Inteligencia Artificial es el de los **sistemas expertos** *o* **sistemas basados en el conocimiento** (**IKBS** *Intelligent Knowledge-Based Systems*). Con ellos se pretende que un programa actúe, ante consultas en un campo específico, como una persona experta en dicho campo. Hay sistemas expertos para diagnóstico médico, reparación y mantenimiento de computadores, concesión de prestamos bancarios, etc.

La **informática gráfica** reúne una serie de materias y conceptos para poder generar y procesar imágenes con objetivos gráficos (edición de documentos, presentación en conferencias, etc.), industriales (diseño de nuevos productos, por ejemplo) o artísticos (realización de cuadros con los movimientos del ratón imitando a un pincel, animación de dibujos, etc.). La informática gráfica ha ideado procedimientos para que estas imágenes se pueden visualizar en dos o tres dimensiones, girarlas, cambiarlas de escala, animarlas (dando la sensación de movimiento), etc. Todas estas operaciones se realizan por el usuario con sencillos lenguajes de órdenes. Las imágenes se pueden generar por medio de programas especiales para dibujar o para pintar (con los que el usuario puede introducir trazos como los de una brocha, pincel o tiza), o por medio de sistemas para producir imágenes digitalizadas (cámaras digitales de vídeo o de fotos, escáner de imágenes, etc.). Esas imágenes después pueden ser procesadas por programas de manipulación de fotos o de animación. Dentro del software gráfico se incluyen paquetes tales como: Adobe Illustrator, Adobe Photoshop, Adobe SuperPaint, CorelDraw, Corel Photo Paint, Fractal Design Painter, Macromedia FreeHand, Micrografx Picture Publisher y PaintShop Pro.

Las **aplicaciones multimedia** son presentaciones producidas con ayuda de computador que se realizan a través de una combinación de texto, gráficos, dibujos, animación, vídeo y sonido. Tradicionalmente las imágenes producidas por computador eran estáticas, con las aplicaciones multimedia se da un salto cualitativo de gran importancia que implica la utilización de configuraciones específicas de equipos informáticos que suelen incluir cámara digital de vídeo, tarjeta de sonido, micrófonos, etc. Además, como las imágenes digitalizadas ocupan mucha memoria, se requieren redes de comunicación de gran ancho de banda, altas capacidades de memoria principal y de disco, y el uso de técnicas de compresión de datos como la definida por el esquema **MPEG** (*Motion Picture Expert Group*). Como una evolución de las aplicaciones multimedia ha surgido el campo de la ***realidad virtual***, que fundamentalmente añade a aquellas una tercera dimensión, y logra la recreación en la pantalla de un computador de lo que aparece en el espacio físico real.

Internet es un sistema mundial de computadores interconectados a través de redes de comunicación. La principal característica de internet es ofrecer un medio, a un precio muy razonable, para poder intercambiar programas, información multimedia (texto, imágenes, vídeo y sonido), y cualquier información almacenable de forma digital, entre cualquier pareja

o conjunto de computadores conectados a la red. Los servicios más utilizados que ofrece internet son los siguientes:

- **Correo electrónico (e-mail)** que permite enviar de forma rápida y eficiente mensajes.
- **Boletines de noticias (news)** sobre temas concretos y distribuidos a grupos de interés en ellos.
- **Acceso remoto** a otro computador (**telnet**), para acceder a los recursos informáticos de éste (ejecutar programas, por ejemplo).
- **Guías para búsquedas** de información (gopher).
- **Charlas interactivas de usuarios en red** (IRC, Internet Relay Chat o **Chats**).
- **Acceso a páginas de la red mundial (www o word wide web)**. Las páginas son documentos con enlaces a otros documentos existentes en el mismo u otros servidores.
- **Comercio electrónico (e-commerce)** que permite realizar transacciones comerciales a través de la red mundial.

El desarrollo de internet está suponiendo una auténtica revolución tanto en cuanto a la forma de difundir y acceder a la información como a la forma de trabajar.

1.9 Conclusión

En este capítulo se ha pretendido dar una visión panorámica del contenido de esta obra. También se ha introducido una serie de definiciones y conceptos para que el lector comprenda qué son, cómo funcionan, cómo se utilizan y para qué sirven los computadores. La mayor parte de estos aspectos serán ampliados en sucesivos capítulos.

Resulta este primer capítulo de interés también porque se ha introducido en una forma comprensible y sencilla, aunque no por ello exenta de rigor, la terminología más usual en informática que se utilizará a lo largo de este libro.

EJERCICIOS

1.1 Codificar las letras del alfabeto utilizando caracteres numéricos. Escribir a continuación, con el código anterior, la frase “En un lugar de la Mancha...”.

1.2 ¿Cuántos bits hay en 32 KB? ¿Y en 64 MB? ¿Y en 4 GB?

1.3 Un computador tiene 36 K palabras de memoria principal y está estructurado en palabras de 32 bits. ¿Cuántos caracteres caben en dicha memoria?

1.4 Suponiendo que su capacidad máxima de memoria es de 32 MBytes, y está organizado en palabras de 32 bits.
a) ¿Cuantos bits tienen en total?
b) ¿Cuál es el ancho (número de hilos) del bus de datos?
c) Suponiendo que el ancho de banda de la memoria es de 200 MBytes/s, indicar la velocidad de transferencia que debe admitir cada hilo del bus de datos.

1.5 Se dispone de una memoria de 4 GBytes, organizada en palabras de 32 bits. Suponiendo que cada hilo transmite a una velocidad de 150 Mbits/segundo, obtener:
a) La anchura del bus de datos y de la memoria.
b) El tiempo necesario para transferir 1 MByte entre la memoria y el procesador.
c) ¿Cuál debe ser la frecuencia de reloj mínima para poder leer una palabra de memoria en un ciclo?

1.6 Suponga que las instrucciones ADD, MUL, LOAD y STORE de un computador, cuyo reloj funciona a una frecuencia de 1,5 GHz, consumen 8, 15, 4 y 4 ciclos de reloj, respectivamente. Obtenga el tiempo que tardan en ejecutarse dichas instrucciones.

1.7 En un catálogo se indica que la velocidad de un computador es de 2 MIPS. Estimar el tiempo aproximado que se tardaría en ejecutar un conjunto de 328.325 instrucciones. ¿Qué inexactitudes presenta dar la velocidad en MIPS?

1.8 Supóngase que se tiene un programa en lenguaje máquina con las siguientes instrucciones:

LLI r0,00
LLI r1,04
ADDS rE,r1,r0

Este programa carga en el registro del procesador r0 el valor 0, en el registro r1 el valor 4, y suma los valores de r0 y r1, llevando el resultado al registro rE. El programa se ejecuta por dos procesadores distintos, A y B, cuyas frecuencias de reloj y número de ciclos que utiliza para ejecutar las instrucciones del programa son las que se dan en la siguiente tabla:

	Procesador A	**Procesador B**
Frecuencia de reloj	1,5 GHz	1 GHz
Nº de ciclos de instrucción LLI	6	3
Nº de ciclos de instrucción ADDS	7	4

a) Obtener los tiempos de ejecución del programa en los procesadores A y B.
b) Obtener los rendimientos (en MIPS) de cada uno de los procesadores para el programa.
c) Discutir la influencia de la frecuencia de reloj y del número de ciclos por cada instrucción en los resultados anteriores.

1.9 Se dispone de un programa que contiene un total de 200 instrucciones máquina, 150 de ellas pertenecientes a un bucle que se ejecuta 50 veces. La ejecución del programa consume un total de N_{TC} = *34.000* ciclos de reloj:
a) ¿Cuál es el número medio de ciclos por instrucción, N_{CI}?
b) ¿Cuál es la frecuencia del reloj del procesador si tarda 34 μs en ejecutarse el programa?

1.10 Un programa de prueba (en lenguaje de alto nivel) se ejecuta en un procesador obteniéndose una velocidad de 100 MIPS y tardando su ejecución 35 segundos. Obtener la frecuencia de reloj sabiendo que por término medio cada instrucción consume 6 ciclos de reloj.

1.11 Para un computador Altos G310 Mk2 (3.6 GHz Intel Pentium 4) se realizan medidas de los tiempos que tardan en ejecutarse los programas del conjunto SPEC para enteros que se indican en la primera columna de la Tabla 1.5, obteniéndose los tiempos que se dan en la columna tercera. En la primera columna se indican los tiempos obtenidos en el computador de referencia. Obtener el valor SPEC para el computador indicado. ¿Qué ocurriría si se utilizase la media aritmética en lugar de la geométrica? (*Solución:* SPECint = 1744).

1.12 Consiga información de al menos cuatro computadores distintos, realice una tabla comparativa de los mismos incluyendo:
a) Marca.
b) Modelo.
c) Microprocesador que utiliza.
d) Longitud de palabra.
e) Velocidad (frecuencia de reloj).
f) Memoria principal, inicial y máxima.
g) Periféricos de que dispone.
h) Nombre del sistema operativo.

Tabla 1.5. Programas citados en el Ejercicio 1.11.

Programa			**Tiempo de referencia**	**Tiempo medido**
Referencia	**Descripción**	**Lenguaje**		
164.gzip	Data compression utility	C	1400	1,16
175.vpr	FPGA circuit placement and routing	C	1400	1,16
176.gcc	C compiler	C	1100	0,538
181.mcf	Minimum cost network flow solver	C	1800	0,909
186.crafty	Chess program	C	1000	0,745
197.parser	Natural language processing	C	1800	1,21
252.eon	Ray tracing	C++	1300	0,578
253.perlbmk	Perl	C	1800	0,843
254.gap	Computational group theory	C	1100	0,565
255.vortex	Object Oriented Database	C	1900	0.620
256.bzip2	compression utility	C	1500	0,116
300.twolf	Place and route simulator	C	3000	0,166

Realice un pequeño informe comparativo.

1.13 Encuentre un ejemplo concreto de cada uno de los tipos de computadores de la Tabla 1.4, incluyendo las características que se indican en dicha tabla (*Sugerencia:* hágalo utilizando la web).

1.14 Obtener a través de la dirección web de los computadores más potentes de la actualidad (TOP500) las siguientes características del computador más potente de la actualidad:
a) Número de procesadores.
b) Modelo de procesador.
c) Capacidad de memoria.
d) Capacidad de disco.
e) Institución propietaria del computador.
f) Fabricante.

1.15 Obtener a través de la dirección web de los computadores más potentes de la actualidad (TOP500, sección *data base*) las siguientes características de los computadores españoles más potentes de la actualidad:
a) Número de procesadores.
b) Modelo de procesador.
c) Capacidad de memoria.
d) Capacidad de disco.
e) Institución propietaria del computador.
f) Fabricante.

1.16 Indicar a qué niveles conceptuales de descripción de una computadora pertenecen los siguientes elementos:
a) La descripción del trazado del interior del circuito integrado que contiene una ALU.
b) Un compilador de C.
c) Un programa en lenguaje máquina.
d) El diseño de un circuito para multiplicar utilizando puertas lógicas.
e) El controlador de un ratón, incluido en un DVD.

1.17 Realice una estimación, para el ejemplo de archivo de la Figura 1.9, de los números de caracteres y bits que ocuparía:
a) cada campo (suponer que cada campo tiene una longitud fija que el lector puede determinar juiciosamente),
b) cada registro, y
c) todo el archivo, suponiendo que contiene en total 13.546 registros.

BIBLIOGRAFÍA

[ACM] ACM, *Computing Reviews* (en todos los números de esta revista aparece la clasificación citada).

[Bee05] Beekman, G.: *Introducción a la Informática;* 6ª Edc. Prentice Hall, 2005.

[Bro03] Brookshear, J. R.: *Computer Science*; 7th Edt. Addison Wesley, 2000.

[Fer98] Fernández, G.: *Conceptos básicos de arquitectura y sistemas operativos;* Sistemas y Servicios de Comunicación, 1998.

[For03] Forouzan, B. A.: *Introducción a la Ciencia de la Computación,* Thomson, 2003.

[Ham03] Hamacher, C.; Vranesic, Z.; Zaky, S.: *Organización de computadores*, McGraw-Hill, 2003.

[Hen00] Hennessy, J. L.; Patterson, D.: *Estructura y diseño de computadores*. Editorial Reverté, 2000.

[Nor03] Norton,P.: *Introduction to Computers,* 5th Edt. MacGraw-Hill, 2003.

[Ort05] Ortega, J.; Anguita, M.; Prieto, A.: *Arquitectura de Computadores*, Thomson, 2005.

[Pat00] Patterson, D. A.; Hennessy, J. L.: *Estructura y diseño de computadores;* Editorial Reverté, 2000.

[Pri05] Prieto, A.; Prieto, B.: *Conceptos de Informática,* Serie Shaum, McGraw-Hill, 2005.

[Tuc94] Tucker, A. B.; Bradley, W. J.; Cupper, R. D.; Garnick, D. K.: *Fundamentos de la Informática*. McGraw-Hill, 1994.

[webTOP] http://www.top500.org/.

CAPÍTULO 2

Tipos y estructuras de datos

Los computadores son máquinas para el tratamiento automático de la información. Esta información no se almacena ni se representa al azar, sino que debe organizarse o estructurarse en forma adecuada para obtener un rendimiento razonable en su memorización, tratamiento y recuperación.

El capítulo comienza estudiando el concepto de dato y de tipo desde un enfoque lógico. A continuación se describen los tipos de datos elementales más usuales en Informática. Por último se analizan algunas estructuras de datos utilizadas en programación, en sistemas operativos y en el diseño físico de computadores.

2.1 Tipos de datos

Se denomina **dato** a *cualquier objeto manipulable por el computador*. Un dato puede ser un carácter leído de un teclado, información almacenada en un disco, un número que se encuentra en memoria principal, etc.

Datos son tanto las **constantes** definidas dentro de los programas, que no alteran su valor durante la ejecución de éstos, como las **variables** utilizadas en dichos programas. Así mismo, son datos la información externa al programa, a la que se puede acceder mediante algún procedimiento, ya esté dicha información grabada en algún medio de memoria masiva o sea generada por algún periférico.

Cuando utilizamos un computador para resolver un problema debemos hacer una abstracción de éste y de las magnitudes que influyen en él. Dichas magnitudes deben ser representadas por *datos*. La representación de una magnitud como un dato, se puede entender como una aplicación que hace corresponder un dato a cada valor de la magnitud. Así, por ejemplo, para resolver un problema en que intervengan distancias, es preciso tener una aplicación, T, del conjunto de valores de la magnitud distancia, **d**, en un conjunto de datos, **D**.

Esta transformación es deseable que esté definida en todo el conjunto de valores de la magnitud. Es conveniente, además, que sea unívoca; es decir, que a dos valores de magnitud distintos les asocie datos distintos. Para que se pueda operar con los datos es necesario que existan unas operaciones internas en el conjunto de datos, que sean semejantes a las operaciones usuales en el conjunto de magnitudes. Dichas operaciones deben cumplir que: la imagen según la transformación $\boldsymbol{T}$ del resultado de una operación en el conjunto de magnitudes

sea igual al resultado de las operaciones correspondientes en el conjunto de datos sobre las imágenes de los operandos.

Si denotamos por + la suma en el conjunto de magnitudes, **d**, lo dicho anteriormente implica que debe existir una operación ⊕ sobre el conjunto de datos, ***D,*** que cumpla:

$$T(x + y) = T(x) \oplus T(y) = X \oplus Y \quad \forall\, x, y \in \mathbf{d}$$

Para que los resultados obtenidos en el conjunto de datos, ***D,*** puedan ser interpretados es necesario que exista una transformación, $\boldsymbol{T^{-1}}$, de estos al conjunto de magnitudes, ***d***. Esta aplicación $\boldsymbol{T^{-1}}$ hará corresponder a cualquier dato, X, un valor de magnitud, x, cuya imagen por ***T*** es el propio dato X.

Se denomina **tipo de dato** al *conjunto de la transformación,* ***T****, y de las operaciones y funciones internas y externas definidas sobre el conjunto de datos*. Distintas transformaciones darán lugar a distintos tipos de datos, aún cuando el conjunto origen a representar sea el mismo.

A continuación se verán *algunos* de los tipos de datos básicos más usuales en Informática: entero, real, carácter, lógico, enumerado y subrango. Nos encontraremos tipos de tipos de datos en la representación de la información, tanto al nivel físico del computador, como en los lenguajes de programación. No todos los tipos de datos existen en todos los lenguajes de programación. Hay lenguajes más ricos que otros en tipos de datos. Concretamente todos los tipos que se van a describir se pueden definir en Pascal.

2.1.1 Datos de tipo entero

El **tipo de datos entero** *es una representación del conjunto de los números enteros*. La representación es posible para un subrango de magnitudes enteras centrado en el origen: números entre $2^{n-1}-1$ y -2^{n-1}. La razón de esta limitación está en la necesidad de utilizar un espacio finito, y fijo, para cada dato, y en el hecho de que la transformación realizada entre los números enteros y el tipo de datos consiste en representar el número en binario y almacenarlo con un número fijo de bits (este proceso se estudiará con más detalle en el capítulo tres).

El número de datos distintos de tipo entero que se pueden generar, *cardinalidad del tipo*, es 2^n (donde n es el número de bits que se utiliza en la representación). Por tanto, si se modifica el número de bits, ***n,*** se obtienen **distintos tipos enteros**. En consecuencia, no todos los números enteros se pueden representar.

A cada operación aritmética sobre el conjunto de los números enteros se le asocia, en el tipo entero, la operación correspondiente. Estas operaciones no pueden realizarse sobre cualquier par de datos enteros, ya que aparecen indeterminaciones debidas a la propia representación. Así si el máximo valor positivo de tipo entero es 32767 (como ocurre si la representación se realiza con $n = 16$), al realizar la operación: 32767 + 1 se obtendría un resultado no representable en el tipo. Este tipo de error se conoce como **desbordamiento** (*overflow*). Cuando esto sucede el computador puede, o no, avisar al usuario. En cualquier caso se obtendrá como resultado algún dato de tipo entero, pero éste no corresponderá con el valor que se obtendría al realizar la misma operación con números enteros. En el caso anterior el número generado sería –32768, si la representación es en complemento a dos. Cualquier operación con datos de tipo entero es exacta salvo que se produzcan desbordamientos. En la Sección 4.5.1 se describe con detalle distintas representaciones de los datos de tipo entero en el interior del computador.

2.1.2 Datos de tipo real

El **tipo de datos real** *es una representación del conjunto de los números reales*. Esencialmente la transformación realizada consiste en expresar el número en la forma:

$$N = M \cdot B^E$$

donde N es el número real a representar, B es la base utilizada para el exponente (que puede estar prefijada para un computador dado), *E* es el exponente del número y *M* es la mantisa. El número se almacena en el computador yuxtaponiendo el signo, el exponente y la mantisa, cada uno con un número de bits prefijado (véase Sección 4.5.2).

Esta representación no permite el almacenamiento de números muy grandes o muy pequeños (del orden de 10^{64} ó 10^{-64}), lo que conlleva que se produzcan **desbordamientos** y **agotamientos** (*overflow* o *underflow*). Por otra parte la limitación del número de bits usados para representar la mantisa provoca una falta de precisión en la representación. Esto es debido a que la aplicación que define al tipo real no es unívoca. Es decir, cada dato de tipo real es la imagen de un conjunto infinito de números reales, concretamente representa a un intervalo de la recta real.

Para los datos de tipo real están definidas las operaciones aritméticas. La suma y la multiplicación de datos de tipo real cumplen la propiedad conmutativa, pero no siempre la asociativa ni la distributiva. Esto sucede porque el orden en que se realizan las operaciones influye en el resultado. En cada operación se producen errores por falta de precisión en la representación (errores de redondeo), que se acumulan durante todo proceso de cálculo (Sección 4.5.2.2).

EJEMPLO 2.1

Se puede poner de manifiesto que no se cumplen las propiedades asociativa y distributiva con el siguiente ejemplo. Supongamos un sistema de representación con base de exponente (B) de 10, una mantisa de 4 dígitos decimales y un exponente de un dígito decimal, y sean:

$$x = 5000 = .5000 \cdot 10^4 \quad y = -4999 = -.4999 \cdot 10^4 \qquad z = 1.500 = .1500 \cdot 10^1$$

Para poder operar estos datos se deben desplazar sus mantisas para que coincidan sus exponentes, lo que puede provocar errores de redondeo:

$$z = .00015 \cdot 10^4 \approx .0001 \cdot 10^4$$

Por la ley asociativa de la suma, $(z+x)+y = .5001 \cdot 10^4 - .4999 \cdot 10^4 = 2.000$ *debería ser igual a*[1] $z+(x+y)$, *que es:*

$$1.500 + (5000 \cdot 10^4 - 4999 \cdot 10^4) = .15 \cdot 10^4 + .1 \cdot 10^1 = .25 \cdot 10^1 = 2.500$$

Por la ley distributiva, $z \cdot (x+y) = 1.500 \cdot 1.000 = 1.500$ *debería ser igual a* $(z \cdot x) \cdot (z \cdot y)$ *que vale:*

$$7500 - 7498 = 2.000$$

En este sistema el dato real positivo más pequeño es $.1000 \cdot 10^{-9}$, *el positivo mayor es* $.9999 \cdot 10^9$, *y la distancia entre dos valores consecutivos es* $.0001 \cdot 10^k$, *siendo k el exponente del dato.*

Los errores de redondeo son especialmente importantes cuando aparecen, directa o indirectamente, cancelaciones. Esto ocurre, por ejemplo, cuando en una expresión se suma una

[1] En los cálculos tenemos que ir igualando exponentes y despreciando a partir de la quinta cifra menos significativa (ya que usamos una mantisa de cuatro dígitos).

cantidad grande, en comparación con el valor de la expresión, lo que hace que se pierdan los dígitos menos significativos, y posteriormente se resta dicha cantidad, u otra del mismo orden de magnitud. Si, con los datos del ejercicio anterior, se le suma y resta x a z queda:

$$(x + z) - x = (.5 \cdot 10^4 + .0001 \cdot 10^4) - 0.5 \cdot 10^4 = .5001 \cdot 10^4 - 0.5 * 10^4 = .0001 \cdot 10^4 = 1 \neq z$$

Usualmente en un computador se pueden utilizar diversas representaciones para los datos reales: simple, doble o cuádruple precisión. En cada uno de estos tipos el número de bits dedicados a representar la mantisa, y a veces también el exponente, es diferente.

2.1.3 Datos de tipo lógico

Los **datos de tipo lógico** *representan valores lógicos o booleanos*. Pueden tomar uno de entre dos valores: verdadero o falso (abreviadamente V, F o 0, 1). Más adelante se expondrá el sistema de representación de estos datos en distintos computadores.

Sobre los valores lógicos pueden actuar los llamados **operadores lógicos**. Los operadores lógicos fundamentales son: Y, O y NO (en inglés: AND, OR y NOT, respectivamente). La definición de las operaciones se hace indicando su resultado para las cuatro posibles combinaciones de valores de los argumentos. La Tabla 2.1 muestra la definición de algunas operaciones lógicas.

En algunos lenguajes de programación hay definidos sobre los datos de tipo lógico otros operadores booleanos, como son: NO-Y, NO-O y NO-exclusivo (en inglés NAND, NOR y XOR). La Tabla 2.1 muestra la descripción de estas funciones.

Tabla 2.1. *Funciones lógicas.*

a	b	$a \wedge b$ (a Y b)	$a \vee b$ (a ó b)	$\overline{a \wedge b}$ (a NoY b)	$\overline{a \vee b}$ (a NoO b)	$a \oplus b$ (a Xor b)	$\overline{a}$ (No a)
0	0	0	0	1	1	0	1
0	1	0	1	1	0	1	1
1	0	0	1	1	0	1	0
1	1	1	1	0	0	0	0

EJEMPLO 2.2

Las siguientes expresiones son relaciones:

1) 7.4 < 3.27 2) 34.7 < 107.3

3) "A" = "J" 4) X + Y ≥ 7.12

El valor lógico resultante de la primera relación es falso. El valor de las tres siguientes es verdadero. La cuarta tendrá valor verdadero si X+Y es mayor o igual a 7.12, en caso contrario será falso.

Un caso particularmente importante de valor de tipo lógico es el obtenido como resultado de una operación de relación sobre datos de un tipo para el que existe una relación de orden

(tipos entero, real, carácter, enumerado y subrango, por ejemplo). Una relación es una expresión formada por dos operandos pertenecientes a un mismo tipo ordenado y un ***operador de relación***. Son operadores de relación los siguientes: ≥ (*mayor o igual*), > (*mayor*), ≤ (*menor o igual*), < (*menor*), = (*igual*), ≠ (*distinto*). El resultado de una operación de relación es el valor lógico **verdadero** si la relación expresada es cierta, y **falso** en caso contrario. Una relación es, pues, una operación externa sobre el tipo de datos de los operandos.

En los capítulos 5 y 6 se consideran, dentro de otro contexto, aspectos complementarios a los vistos en esta sección acerca de variables y funciones lógicas.

2.1.4 Datos de tipo carácter

El **tipo de datos carácter** *representa elementos individuales de conjuntos finitos y ordenados de caracteres.* El conjunto de caracteres representado depende del computador. Uno de los conjuntos más usuales es el ASCII (véase Apéndice A).

No hay ninguna operación interna sobre datos de tipo carácter (salvo la asignación, A="B"). Normalmente existen funciones de conversión de tipo. Como por ejemplo la que asocia a cada dato de tipo carácter un valor entero, que indica su posición en el código.

EJEMPLO 2.3

Supongamos que **ORD** *es la función que devuelve el código de un carácter y* **CHR** *es la función que da como resultado un carácter aceptando como argumento el valor de código. Si el código usado es el ASCII se tendrá que:*

ORD*("A") es 65*   **CHR***(65) es "A"*   **CHR(ORD***("B")***)** *es "B"*

Nótese que los valores constantes de tipo carácter se escriben entre comillas, para distinguirlos de nombres de variables.

2.1.5 Datos de tipo enumerado

El **tipo de datos enumerado** *se define explícitamente dando un conjunto finito de valores.* Al contrario de los tipos vistos anteriormente, el tipo de datos enumerado, como el tipo subrango y los tipos estructurados que veremos en las secciones siguientes, no es un tipo normalizado. Puede haber muchos tipos de datos enumerados distintos dentro de un programa en un lenguaje determinado, mientras que no habrá más que un tipo lógico. En cierto modo podemos considerar al tipo de dato enumerado como una clase de tipos de datos, a la que pertenecerán todos los tipos definidos por enumeración.

Los tipos de datos vistos en las secciones anteriores son usualmente tratados por el computador a nivel hardware. Mientras que el tipo de datos enumerado y el tipo de datos subrango sólo son interpretados por el software (por el traductor del lenguaje).

Internamente los datos de tipo enumerado se almacenan como valores enteros. A cada valor del tipo se le asocia un entero consecutivo, comenzando por cero. Existen, como en el tipo carácter, funciones de conversión a entero. Pueden también existir funciones que generen el valor siguiente (sucesor) o anterior (predecesor) a uno dado, según el orden en que éstos se definieron.

EJEMPLO 2.4

Las siguientes sentencias Pascal definen dos tipos de datos enumerados y declaran variables de dichos tipos:

TYPE *día = (lunes, martes, miércoles, jueves, viernes, sábado, domingo);*
color = (azul, rojo, verde, amarillo);
VAR *libre, ocupado : día;*
pared : color;

Día *y* ***color*** *son tipos de datos enumerados.* ***Libre*** *y* ***ocupado*** *son dos variables del tipo enumerado día. El valor de libre será siempre uno de los dados en la definición de día.*

En Pascal están definidas las funciones internas **predecesor** *(***PRED***) y* **sucesor** *(***SUCC***), que dan los valores anterior y siguiente a uno dado. En los tipos día y color:*

PRED*(jueves) es miércoles*

SUCC*(rojo) es verde*

SUCC ***(*****PRED***(verde)****)*** *es verde*

2.1.6 Datos de tipo subrango

El **tipo de datos subrango** *se define a partir del tipo entero, carácter o de un tipo enumerado*. Un dato de tipo subrango puede tomar determinados valores del tipo original, a partir del cual se ha definido el subrango, entre un mínimo y un máximo. Con datos de tipo subrango se pueden realizar las operaciones definidas para el tipo original.

EJEMPLO 2.5

Las siguientes sentencias Pascal definen tipos subrango:

TYPE *dígito = 0...9;*
letra = "a"..."z";
laborable = lunes...viernes;

Dígito es un tipo subrango definido a partir del tipo entero. Los datos de tipo dígito pueden tomar los siguientes valores: 0,1,2,3,4,5,6,7,8,9.

2.2 Estructuras de datos

Los tipos de datos vistos en el apartado anterior (Sección 2.1) se suelen denominar elementales, y se pueden utilizar para construir tipos de datos más elaborados. Una **estructura de datos** o **tipo de dato estructurado** *es un tipo de dato construido a partir de otros tipos de datos.* Así un dato de tipo complejo, que representa al conjunto de los números complejos, es un par ordenado de datos reales, y por tanto un tipo de dato estructurado.

Un dato de un tipo estructurado está compuesto por una serie de componentes y alguna relación existente entre ellos. Normalmente la relación suele ser de orden aunque puede ser de otro tipo. Los datos componentes pueden ser elementales u otra estructura de datos.

Una estructura de datos se dice que es ***homogénea*** cuando todos los datos elementales que la forman son del mismo tipo. En caso contrario se dice que la estructura es ***heterogénea***. Por ejemplo, el tipo de datos complejo es una estructura homogénea, tanto la parte real como la parte imaginaria se representan con datos reales.

Siempre que se utilice un dato en un programa debe estar determinado su tipo, para que el traductor sepa cómo debe tratarlo y almacenarlo. En el caso de datos de tipos elementales, el tipo del dato determina el espacio que se usa en memoria. Esto puede no ocurrir si el dato es de un tipo estructurado. Algunos tipos estructurados (listas, grafos y árboles) se declaran sin especificar el número de componentes que van a tener. En este caso el compilador les reserva el espacio de memoria mínimo que necesitan. Durante la ejecución del programa la estructura de datos puede ir creciendo; es decir, ocupando más memoria. En cualquier caso el máximo espacio al que pueden llegar está limitado por el espacio libre en el programa. Si se necesitase más memoria de la disponible en el programa, éste terminaría por error. Una estructura de datos que es gestionada de esta forma se dice que es ***dinámica***, ya que la memoria que necesita se asigna dinámicamente. Por el contrario, una estructura de datos que siempre ocupa el mismo espacio se dice que es ***estática***. Los tipos de datos más utilizados son: arrays, cadenas de caracteres, registros, listas, árboles y grafos.

2.2.1 Arrays (Matrices o Formaciones)

El array (también llamado **formación** o **matriz**) es la estructura de datos más usual. Existe en todos los lenguajes de programación y en algunos es de las pocas estructuras de datos existente (Basic y FORTRAN). Por otra parte, la memoria de un computador puede considerarse como una matriz lineal.

Un **array** *es una estructura de datos formada por una cantidad fija de datos de un mismo tipo, cada uno de los cuales tiene asociado uno o más* **índices** *que determinan de forma unívoca la posición del dato en el array*. Cada índice es un dato de tipo subrango. Para cada combinación posible de valores de índices existe un y sólo un dato del tipo constituyente, o **elemento** del array.

Podemos imaginar un array como una estructura de celdas donde se pueden almacenar valores. La Figura 2.1 representa una matriz de un solo índice que toma valores de 1 a 7.

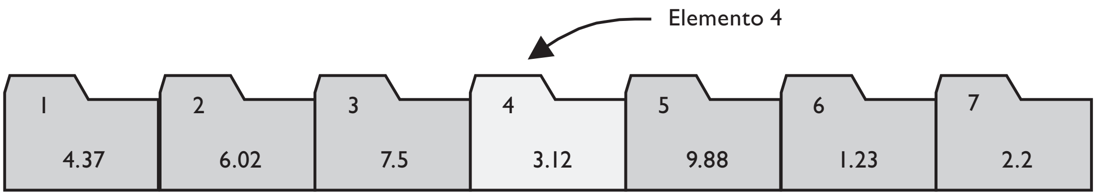


Figura 2.1. *Representación de un array lineal de 7 elementos.*

El array de la Figura 2.2 utiliza dos índices con valores entre 1 y 3, el primero, y entre 1 y 5, el segundo. Cada elemento de esta matriz está especificado por un par ordenado de números, el valor de los dos índices.

Al número de índices del array se le denomina **número de dimensiones** del array. La **dimensión** de la formación está dada por los valores máximos de los índices, y el número total de elementos es el producto de estos valores máximos. En los ejemplos de la Figuras 2.1 y 2.2, el número de dimensiones de los arrays son 1 y 2, las dimensiones son (7) y (3; 5), y los números totales de elementos son 7 y 15, respectivamente.

Un array puede considerarse como una aplicación de un dominio subrango (o producto cartesiano de subrangos) al tipo de datos del array.

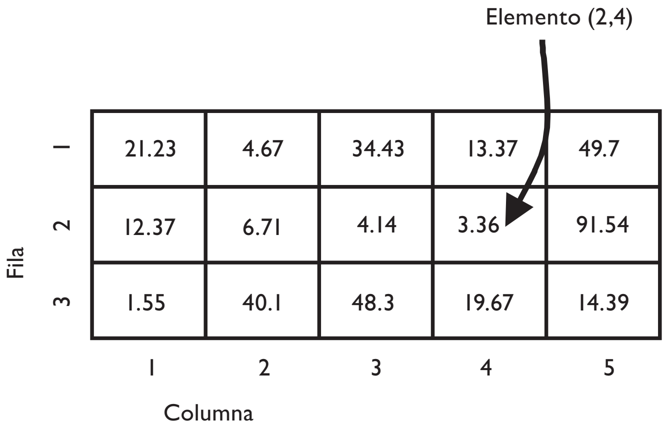


Figura 2.2. *Representación de un array rectangular (es decir, de dos dimensiones). El elemento 2,4 contiene el valor 3.36.*

EJEMPLO 2.6

Las siguientes sentencias FORTRAN declaran que IESC es un array de 30 elementos reales, direccionados por dos índices, el primero en el subgrupo 1...10, y el segundo en el subrango 1...3.

REAL *IESC*
DIMENSIÓN *IESC(1:10,1:3)*
La declaración en **C** *de este array es:*
float *IESC[10][3];*
El mismo array se declara en Pascal usando las siguientes sentencias:
var *IESC* **: array***[1...10,1...3]* **of real;**

Al seleccionar el elemento IESC(1,3) estamos especificando un elemento del array, que es un dato de tipo real. Por tanto se podrá realizar con IESC(1,3) cualquier operación aritmética.

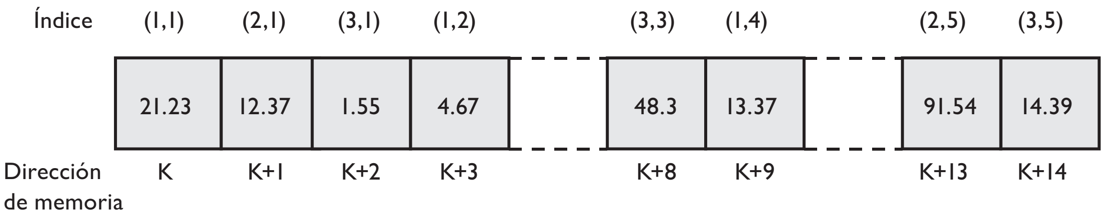


Figura 2.3. *Almacenamiento en memoria del array bidimensional de la Figura 2.2. (Suponiendo almacenamiento por columnas.)*

La principal operación que se puede realizar con arrays es la ***selección***. La selección consiste en especificar un elemento determinado de la array. Esta operación se efectúa dando un valor para todos y cada uno de los índices del array. Con el elemento seleccionado se

pueden realizar las operaciones propias de su tipo. Así con cada elemento de un array real, una vez seleccionado, se pueden realizar las operaciones definidas para datos del tipo real (Sección 2.1.2).

Los arrays son, normalmente, estructuras de datos estáticas. Esto es, al definirlas se especifica el número de elementos que las constituyen. Este dato lo utiliza el compilador para reservar el espacio necesario para almacenarla. Las matrices se almacenan en memoria ocupando un área contigua. Cada elemento ocupa el mismo número de palabras, que será el que corresponda al tipo de éstos. Los elementos se colocan en la memoria según un orden prefijado de los índices. Un **array dinámico** es un array cuyo tamaño puede cambiarse durante la ejecución del programa.

EJEMPLO 2.7

Por lo general las matrices se almacenan por columnas. La matriz IESC del Ejemplo 2.6 se grabará en memoria según la disposición que indica la Figura 2.3.

Si un array está dimensionado, por ejemplo, como $A(F_X, C_X)$*, el traductor se encarga de generar el código máquina para, a partir de los subíndices (I,J) de los elementos referenciados obtener la posición en memoria de dicho elemento. La posición, p, relativa dentro de la zona reservada para el array se puede obtener con la siguiente expresión:*

$$p = F_X \cdot (J-1) + I$$

2.2.2 Cadenas de caracteres

Una **cadena de caracteres** (*string*) *es una estructura de datos formada por una secuencia de caracteres*. En una variable de tipo cadena se puede almacenar una palabra, una frase, una matrícula de un coche, una temperatura, etc. Las constantes de este tipo se escriben normalmente entre comillas.

EJEMPLO 2.8

Para declarar la variable TEXTO como cadena de 30 caracteres en UCSD Pascal se utilizaría la siguiente sentencia:

var *TEXTO* **: string***[30]***;**

En C la definición de esta variable es: **char** *TEXTO[30].*

Sobre datos de tipo cadena de caracteres se pueden realizar las siguientes operaciones:

- **Concatenación:** Consiste en formar una cadena a partir de dos ya existentes, yuxtaponiendo los caracteres de ambas. Si se denota por + al operador **concatenación**, el resultado de "ab"+"cd" es "abcd".
- **Extracción de subcadena:** Permite formar una cadena a partir de otra ya existente. La subcadena se forma tomando un tramo consecutivo de la cadena inicial. Si NOMBRE es una variable de tipo cadena que contiene "PEDRO ORTEGA" y denotamos

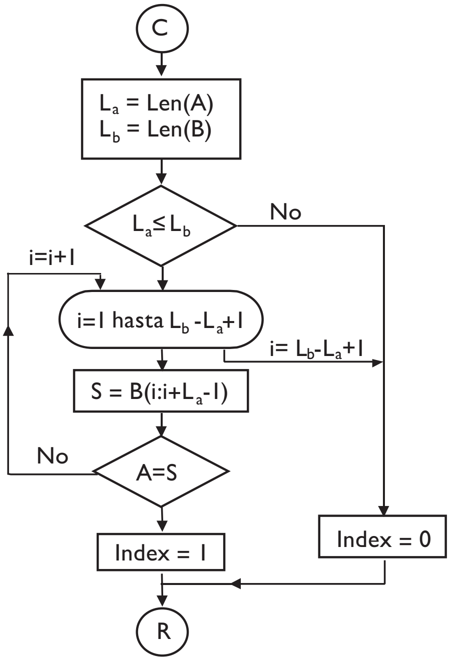
por ***(n:m)*** la extracción de los caracteres ***n*** a ***m***, entonces NOMBRE (1:5) es una subcadena que contiene "PEDRO".

- **Comparación de cadenas:** Es posible comparar dos cadenas en conjunto. Se considera menor aquélla en que el primer carácter en que difieren ambas es menor.
- **Obtención de longitud:** La longitud de una cadena es un dato de tipo entero, cuyo valor es el número de caracteres que contiene.

EJEMPLO 2.9

En FORTRAN77 existe la función **index***, que permite determinar si el valor de una cadena (A) se puede obtener como subcadena de otra cadena (B). Esto permite conocer por ejemplo, si una palabra aparece en una frase. En el caso de que aparezca la cadena A en la cadena B, la función devuelve la posición donde comienza. Esta función se puede realizar con las operaciones descritas en esta sección, tal como se muestra en la Figura 2.4.*

Aunque una cadena de caracteres y un array lineal de caracteres puedan contener la misma información, son tipos de datos distintos, como se ha podido apreciar. Representan objetos distintos y permiten realizar operaciones diferentes.



	Nombre	Variables Tipo	Contenido
Entrada	A	Cadena	Palabra buscada
	B	Cadena	Frase
Salida	Index	Entero	Posición de A en B
Auxiliares	S	Cadena	Cadena auxiliar
	I	Entero	Contador
	L_a	Entero	Longitud de A
	L_b	Entero	Longitud de B

Figura 2.4. *Organigrama de la función* ***index****.*

2.2.3 Registros

Un **registro** *es una estructura de datos formada por yuxtaposición de elementos que contienen información relativa a un mismo ente*. A los elementos que componen el registro se les denominan **campos**. Cada campo es de un determinado tipo. Los campos dentro del

registro aparecen en un orden determinado, y se identifican por un nombre. Para definir un registro es necesario especificar el nombre y tipo de cada campo. Los campos pueden ser de un tipo estructurado.

EJEMPLO 2.10

Para representar la información de nómina de un empleado se puede utilizar un registro con los siguientes campos: apellido y nombre, edad, número de trienios, número de hijos, estado, empleo. La declaración de este registro en COBOL es:

```
01 EMPLEADO
02 NOMBRE      PIC A(36)
02 EDAD        PIC 99
02 TRIENIOS    PIC 99
02 HIJOS       PIC 99
02 ESTADO      PIC A
02 EMPLEO      PIC X(24)
```

"Empleado" es el nombre del registro. En las líneas siguientes aparecen los nombres de campos (después de ***02****). De cada campo se da su tipo (o descripción, en inglés* ***PIC****TURE), que puede ser* ***A****(alfabético),* **9***(numérico), o* ***X****(alfanumérico).*

Con los registros se pueden realizar asignaciones del registro completo a una variable de tipo registro, definida con los mismos campos en el mismo orden. Se puede realizar, al igual que en arrays, la selección de un campo. Esto se realiza especificando el nombre del campo.

2.2.4 Listas

Una **lista** *está formada por un número variable de datos* (**elementos**) *de un mismo tipo, ordenados según una secuencia lineal*. Cada elemento, salvo el primero, tiene un predecesor en la lista. Todos los elementos, salvo el último, tienen un sucesor. La lista es una estructura dinámica. Se pueden representar mediante una lista: los números de las páginas de un libro en que aparece una palabra, los componentes de una máquina, el itinerario de un autobús, las letras de un abecedario, la cola de espera de una carnicería.

Formalmente podemos definir una **lista** como *una estructura de datos formada por registros de al menos dos campos, en que uno de ellos contiene información que permite localizar al siguiente registro en la lista según una secuencia dada.*

Con una lista se pueden realizar las siguientes operaciones:

- **Añadir un elemento:** Esto puede realizarse al final de la lista, al principio, o entre dos elementos *(inserción).*
- **Eliminar un elemento:** Al eliminar un elemento no se pierde la secuencia lógica. Esto es, el predecesor del elemento eliminado pasa a ser el predecesor del elemento siguiente al eliminado.
- **Acceder al primer elemento de la lista:** El primer elemento es normalmente el único al que se puede acceder directamente.

- **Acceder al elemento siguiente del último procesado:** Éste es el mecanismo normal de acceso a la lista. Al acceder a un elemento éste no se elimina. La lista se puede leer desde el comienzo tantas veces como sea necesario.
- **Saber si la lista está vacía:** La lista está vacía si no contiene ningún elemento.

Aunque a primera vista pueda parecer que una lista es semejante a un array lineal, éste es una estructura completamente diferente. Por un lado la lista es una estructura dinámica, ocupa en memoria el espacio necesario para albergar los elementos que se le han añadido. Por otro lado la lista no es direccionable, tan sólo se puede recuperar un elemento accediendo previamente a los que le anteceden, y por tanto en cada momento hay sólo un elemento en disposición de ser procesado.

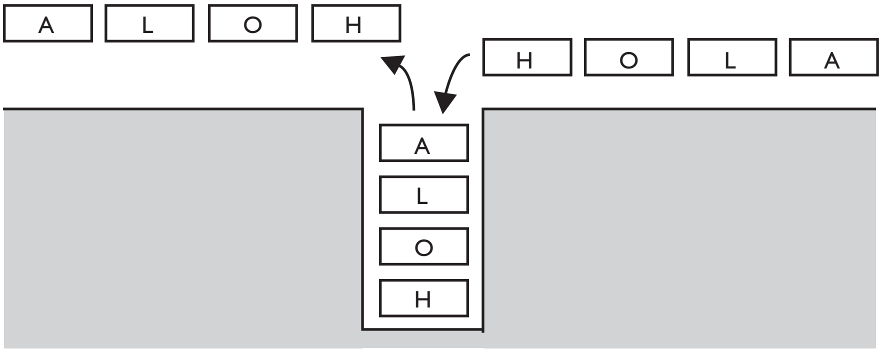


Figura 2.5. *Pila de caracteres. Las letras salen en el orden inverso al que entran.*

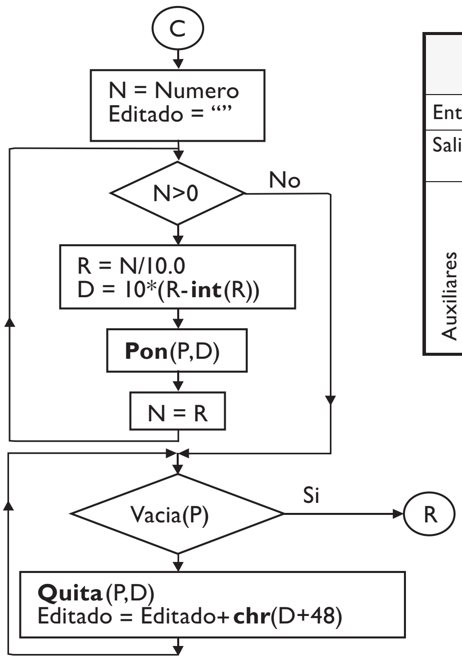
Un caso particular de lista especialmente importante es aquel en que se añaden y eliminan elementos sólo en un extremo. Se denomina **pila** o ***lista LIFO*** (del inglés ***L****ast* ***i****nput* ***f****irst* ***o****utput”:* “el último que entró es el primero en salir) a una lista en que las inserciones y eliminaciones se realizan sólo al principio de la lista. Es decir, cualquier elemento añadido pasa a ser el primero de la lista. Además no se puede eliminar más que el elemento que ocupa el primer lugar de la lista en ese momento. Una pila se corresponde con la idea intuitiva de una pila de objetos (véase la Figura 2.5), en que el último objeto colocado es el primero que se retira. Las pilas se utilizan en hardware y software para almacenar las direcciones de instrucciones desde las que se hacen llamadas a subrutinas.

EJEMPLO 2.11

Si se desea convertir una cantidad entera y positiva N en su representación decimal en forma de cadena de caracteres (lo que se denomina normalmente “editar” o pasar de la representación interna del número a representación binaria según el código de E/S: ASCII, por ejemplo), se puede seguir el siguiente procedimiento:

- *Extraer los valores numéricos de cada dígito individual dividiendo sucesivamente por 10. Los dígitos así se obtienen a partir del menos significativo. Una vez extraído cada dígito se introduce en una pila (“apilar”).*
- *Tomar los dígitos almacenados en la pila (“desapilar”), y añadirlos a la cadena de caracteres.*

La Figura 2.6 es un organigrama que detalla este procedimiento. En dicho organigrama **Pon** *es una sentencia que añade un valor a la pila (“apila”) y* **Quita** *elimina (“desapila”) un elemento de la pila.* **Vacia** *es una función lógica que es verdad si la pila está vacía.*



Variables			
	Nombre	Tipo	Contenido
Entrada	Numero	Entero	Número a editar
Salida	Editado	Cadena	Representación de Número
Auxiliares	N	Entero	Parte de Numero sin procesar
	R	Real	Parte más significativa de N
	D	Entero	Dígito
	P	Pila de enteros	Dígitos de numero inicialmente está vacía

Funciones externas	
Nombre	Descripción
int	Devuelve la parte entera de un número real
chr	Devuelve el carácter cuyo código ascii dado como argumento
Pon	Añade un elemento a la pila
Quita	Obtiene, y retira, un elemento de la pila

Figura 2.6. *Organigrama del proceso descrito en el Ejemplo 2.11.*

Se denomina **cola** a una lista en que las inserciones se realizan sólo en el final y sólo se puede acceder o eliminar en un instante dado el primer elemento de la lista. Las colas se suelen denominar también **listas FIFO** (del inglés: “First In, First Out”, “El primero que entra es el primero que sale”). Las colas, entre otros lugares, se usan en sistemas operativos para la gestión de trabajos no interactivos (procesamiento por lotes o “batch”).

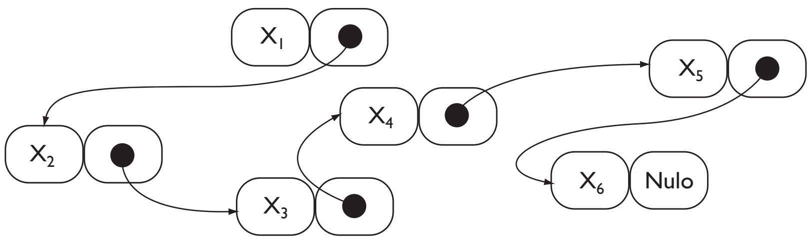


Figura 2.7. *Lista encadenada en memoria.*

Las listas se memorizan utilizando punteros. Un **puntero** *es un dato que contiene una dirección de memoria.* Para cada elemento de la lista se almacena junto a su contenido un puntero que contiene la dirección del elemento siguiente. La Figura 2.7 muestra esquemáticamente cómo se puede almacenar una lista en memoria.

2.2.5 Árboles

Un **árbol** *es una estructura de datos formada por elementos del mismo tipo, llamados* ***nodos****, relacionados de tal modo que el árbol puede descomponerse en:*

- *un nodo, llamado* ***raíz****.*
- *un conjunto finito de objetos de tipo árbol, llamados* ***subárboles*** *del nodo raíz.*

En la Figura 2.8 se muestra la representación gráfica de un árbol.

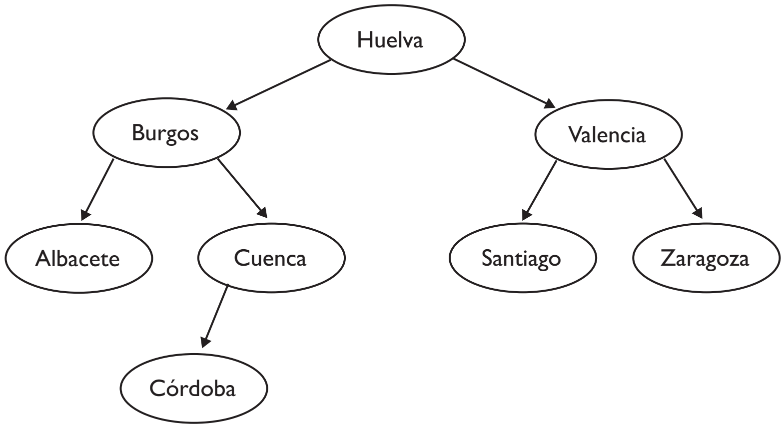


Figura 2.8. *Esquema de un árbol.*

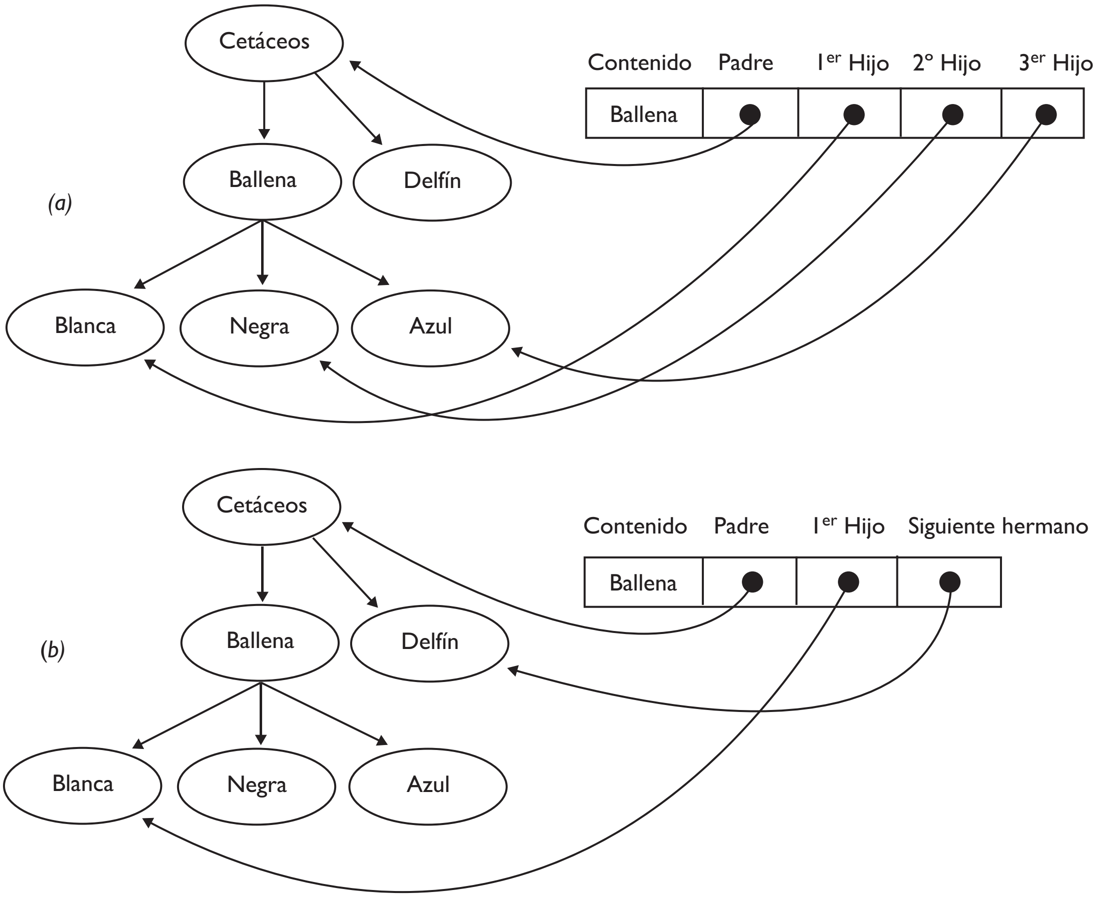


Figura 2.9. *Representación en memoria de un nodo de un árbol: (a) con punteros padre-hijo; (b) con punteros entre hermanos.*

Se denomina **hijo** de un nodo a cada uno de los nodos que dependen de él. En la Figura 2.8 los nodos "Burgos" y "Valencia" son hijos del nodo "Huelva". Se dice también que "Huelva" es el **padre** de estos nodos. Se denomina **grado** de un nodo al número de subárboles que sustenta. En la Figura 2.8 el grado de "Huelva", "Burgos" y "Valencia" es dos, el de "Cuenca" es uno, y el de los demás es cero. El ***orden*** de un árbol es el mayor de los grados de sus nodos. El árbol de la Figura 2.8 es un árbol de orden dos o binario.

Un árbol es una estructura dinámica. Su representación en el interior de un computador se realiza utilizando punteros. Cada nodo puede incluir varios punteros: uno para dirigirse al padre, y cada uno de los restantes para dirigirse a cada uno de los hijos. Esto permite moverse con gran facilidad dentro del árbol en cualquier dirección (hacia arriba o hacia abajo), pero presenta el inconveniente de que el número de punteros para cada nodo no es fijo y puede ser excesivamente grande (véase Figura 2.9a). Normalmente se utiliza otra estructura con sólo tres punteros por nodo (Figura 2.9b), uno para el padre, otro para el primer hijo, y el tercero para el siguiente hermano.

Frecuentemente, los procesos que se realizan con un árbol suelen implicar realizar un recorrido del mismo. En un recorrido visitamos los nodos por orden, siguiendo algún criterio preestablecido (por ejemplo, visitar la raíz, el subárbol izquierdo y después el subárbol derecho es hacer un recorrido en preorden).

Dirección	Información del nodo	Punteros padre	1er hijo	2º hijo
1	Huelva	nulo	2	5
2	Burgos	1	4	7
3	Santiago	5	nulo	nulo
4	Albacete	2	nulo	nulo
5	Valencia	1	3	8
6	Córdoba	7	nulo	nulo
7	Cuenca	2	6	nulo
8	Zaragoza	5	nulo	nulo

Figura 2.10. *Representación en memoria del árbol de la Figura 2.8.*

EJEMPLO 2.12

El árbol de la Figura 2.8 se puede representar en memoria de un computador tal como se muestra en la Figura 2.10.

2.2.6 Grafos

Un **grafo** *es una estructura de datos formada por elementos del mismo tipo, llamados* ***nodos****, y relaciones entre ellos, denominadas* ***arcos***. Los arcos, que suelen estar etiquetados, representan conexiones entre los nodos. Los grafos son estructuras dinámicas que pueden

almacenarse siguiendo una sistemática semejante a la usada para los árboles. Conceptualmente son más generales que las listas y los árboles, ya que estos pueden verse como casos particulares de grafos. En un grafo es posible buscar caminos mínimos, detectar conectividad y simplificación, al igual que en los árboles, estos procesos suelen implicar realizar un recorrido del grafo.

EJEMPLO 2.13

Un mapa de carreteras puede representarse mediante un grafo en el que cada nodo representa una población, o un cruce de carreteras, y cada arco una carretera. Los arcos pueden etiquetarse con la longitud de la carretera. La Figura 2.11 muestra el grafo de un mapa de carreteras. Sobre él se puede comprobar que el camino más corto para ir de Bailén a Antequera es el que pasa por Jaén.

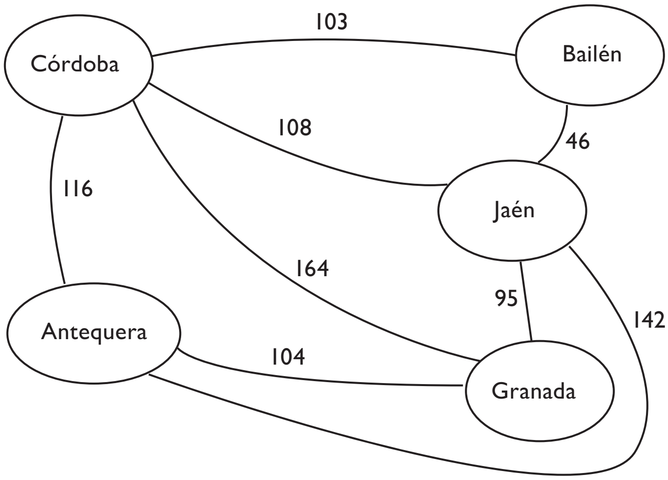


Figura 2.11. *Grafo de carreteras.*

2.3 Conclusión

En este capítulo se ha abordado el estudio de los datos elementales y de las estructuras de datos. Los datos se utilizan para representar elementos de información en los sistemas informáticos, tanto a nivel del hardware como en los programas que se ejecutan sobre él.

En los lenguajes de programación nos encontraremos algunas de estas estructuras de datos y elementos para construir otras.

De los datos elementales y de las estructuras de datos interesa no sólo la forma en que pueden ser almacenados, si no también las operaciones que se pueden realizar sobre ellos. La elección de la estructura de datos a utilizar en cada caso dependerá esencialmente de la facilidad para realizar las operaciones necesarias, y condicionará la eficiencia de la solución adoptada. En el capítulo siguiente se aborda el estudio de los algoritmos y de las técnicas para diseñarlos, en el pueden encontrarse más ejemplos de diseño de estructuras de datos.

EJERCICIOS

2.1 Se desean calcular las siguientes expresiones:
a) Z + X – Y * V
b) Z * (X + Z) / Y
c) X / Y + Z / Y – Y / X
donde X, Y, Z y V son datos de tipo real, en un computador que utiliza una representación en coma flotante de 7 bits de mantisa y 4 de exponente. Si los valores de las variables son de los siguientes órdenes de magnitud:

X ≈ 103 Y ≈ 103 Z ≈ 10–1 V ≈ 1

¿En qué orden se deben realizar las operaciones en cada caso? Comprobar el resultado dando valores a las variables. Repetir el ejercicio para el caso en que X y Z sean datos de tipo entero, representados en el computador con 12 bits, y que Z es del orden de 1.

2.2 Decir en qué casos son ciertas las siguientes expresiones:
a) X o (3 > Y)
b) («A» = T) y (Z < R)
c) D < Sábado
d) (L = CHR(64)) y (ORD(L) > 32)

¿De qué tipos deben ser las variables involucradas, para que dichas expresiones sean correctas?

2.3 ¿Qué estructuras de datos son más apropiadas para representar la siguiente información?
a) Los nombres de los distintos dueños que ha tenido una parcela.
b) Los movimientos de una cuenta bancaria.
c) El horario de salida de autobuses de una estación.

2.4 En bastantes lenguajes de programación no existen estructuras de datos dinámicas. Sin embargo, se pueden utilizar arrays para simular dichas estructuras (¿qué limitación nos impone la utilización de arrays?). Indicar cómo se pueden simular en estos lenguajes las estructuras Pila, Lista y Arbol.

2.5 ¿Cuánto ocupa en memoria el árbol de la Figura 2.8, si se utilizan punteros padre-hijo, cada nodo contiene 6 bytes de información y cada puntero se almacena en 2 bytes? ¿Y si se usan punteros entre hermanos?

2.6 Describe un método para encontrar el camino mínimo entre dos nodos de un grafo.

2.7 Un árbol binario puede usarse para buscar elementos. Cada elemento se almacena en un nodo de tal modo que sea mayor que el contenido de cada uno de los nodos del subárbol izquierdo y menor que los del subárbol derecho (como en el árbol de la Figura 2.9). Piensa cómo se puede realizar una búsqueda, para saber si un determinado valor está en el árbol.

2.8 ¿Como podrían representarse los números racionales? ¿Qué ventajas e inconvenientes tiene trabajar con esta representación respecto a la utilización de números reales?

2.9 Diseña una estructura de datos para almacenar un grafo.

2.10 En algunas calculadoras de bolsillo se utiliza una pila en lugar de paréntesis. En estas calculadoras existen dos teclas una para guardar en la pila el valor actual y otra para recuperar un valor de la pila. Explica cómo se puede usar este mecanismo para sustituir a los paréntesis en la evaluación de una expresión.

BIBLIOGRAFÍA

[Goo77] Goodman, S. E.; Hedetniemi, S. T.: *Introduction to the desing and analysis of algorithms*. McGraw-Hill, Kogakusha, 1977.

[Joy03] Joyanes, L.: *Fundamentos de programación*. 3ª Ed. McGraw-Hill, 2003.

[Joy03a] Joyanes, L.; Rodríguez, L.; Fernández, M.: *Fundamentos de programación. Libro de problemas. Algoritmos, Estructuras de datos.* 2ª Ed. McGraw-Hill, 2003.

[Lew85] Lewis, T. G.; Smith, M. Z.: *Estructuras de datos, programación y aplicaciones*. Paraninfo, 1985.

[Mar03] Martí, N.; Ortega Y.; Verdejo, J. A.*: Estructuras de datos y algoritmos.. Ejercicios y problemas resueltos*. Pearson Prentice Hall, 2003

[Pri05] Prieto, A.; Prieto, B.: *Conceptos de Informática,* Serie Shaum, McGraw-Hill, 2005.

[Wir87] Wirth, N.: *Algoritmos y Estructuras de datos*. Prentice Hall, 1987.

CAPÍTULO 3

Algoritmos

El tratamiento de la información se debe realizar de un modo sistemático. La resolución de cualquier problema conlleva el encontrar un método de resolución expresado con la suficiente precisión para poder ser descompuesto en acciones realizables por el computador (esto es lo que denominaremos un *algoritmo*).

El capítulo comienza estudiando el concepto de algoritmo, presentando distintas técnicas de representación. Estos conceptos se utilizarán para expresar los distintos algoritmos que se presenten en los siguientes capítulos.

Si bien los conceptos introducidos aquí son fundamentales para la realización de programas, no debe abordarse su lectura como un manual de programación.

En el Capítulo 17 se da una visión completa del proceso de desarrollo de software.

3.1 Algoritmos

Un computador es capaz de realizar determinadas acciones sencillas, tales como sumar, restar o transferir datos. Estas acciones son de por sí útiles; sin embargo, los problemas que normalmente interesa resolver son más complejos. Para solucionar un problema real es necesario encontrar un método de resolución del problema y posteriormente descomponerlo en acciones sencillas, que el computador sea capaz de realizar.

3.1.1 Concepto de algoritmo

No todos los métodos de solución de un problema son susceptibles de ser utilizados por un computador. Para que un procedimiento pueda ser implantado en un computador, o en otra máquina capaz de interpretar instrucciones, debe cumplir determinados requisitos:

- El procedimiento debe estar compuesto de *acciones bien definidas*, esto es, no ambiguas. El significado de cada acción debe ser único, en el contexto en que aparece.
- El procedimiento debe estar formado por una *secuencia finita de operaciones*. Además debe quedar perfectamente definido el orden en que se van a realizar las instrucciones.
- Por último, el procedimiento debe acabar en un *tiempo finito*. Un procedimiento que puede no acabar nunca no es útil para resolver un problema.

Un procedimiento o método de solución, para resolver un problema, que cumpla estos requisitos se dice que es un **algoritmo** que resuelve ese problema.

Se puede dar la siguiente definición de algoritmo [Goo77]: Un **algoritmo** *es un procedimiento no ambiguo que resuelve un problema. Un procedimiento es una secuencia de operaciones bien definidas, cada una de las cuales requiere una cantidad finita de memoria y se realiza en un tiempo finito.*

EJEMPLO 3.1

Supóngase que se quiere explicar a alguien lo que debe hacer para determinar si un número es par. Se le puede decir:

Si el número se puede obtener sumando doses, es par. Si para construirlo hay que sumar a uno cualquier secuencia de doses es impar.

Se entiende perfectamente lo que se quiere decir. No obstante, este enunciado no es un algoritmo, pues no constituye una secuencia de operaciones. Además, presenta ambigüedad. Este mismo enunciado se podría expresar en una forma más precisa, del siguiente modo:

1. **Leer** N
2. **Si** $N = 2$ **entonces Escribe** *("es par")*
3. **Si** $N = 1$ **entonces Escribe** *("es impar")*
4. $N = N - 2$
5. **Si** $N > 0$ **ir a** *2*
6. **Fin**

Ahora se ha definido una secuencia de operaciones. Para que esta secuencia sea un algoritmo, las operaciones que aparecen en ella han de ser no ambiguas y para cualquier dato de entrada el proceso debe acabar en un tiempo finito.

Es suficiente una breve inspección del procedimiento para comprobar que este acaba después de (N + 1)/2 iteraciones. Por tanto, este procedimiento es un algoritmo.

Para resolver un mismo problema, se pueden definir infinidad de algoritmos. Es posible realizar comparaciones entre algoritmos que resuelvan un mismo problema. Normalmente interesa, no sólo encontrar un algoritmo, sino que éste sea suficientemente "bueno". La bondad de un algoritmo se mide, básicamente, por dos factores:

- **El tiempo** que se necesita para ejecutarlo. Para comparar dos algoritmos no es necesario conocer el tiempo real que invertirá el computador, basta con saber el número de instrucciones de cada tipo necesarias para resolver el problema. Con frecuencia, el tiempo empleado es función del tamaño del problema, aumentando con éste. En estos casos, se debe analizar no sólo el tiempo empleado en resolver un problema de tamaño dado, si no la velocidad con la que aumenta el tiempo al aumentar el tamaño del problema.
- **Los recursos** que se necesitan para implantar el algoritmo. Concretamente en el caso de que el algoritmo lo deba ejecutar un computador, se usan fundamentalmente los siguientes recursos: memoria principal para almacenar los datos y las instrucciones, y memoria masiva para almacenar datos auxiliares.

No hay ningún procedimiento riguroso que permita construir un algoritmo que resuelva un problema dado. El **diseño de algoritmos**, como toda tarea creativa, es una tarea compleja, en la que se pueden seguir pocas normas, teniendo por tanto gran importancia la imaginación y experiencia de la persona que lo realiza. Tan sólo se pueden dar normas generales y herramientas que pueden ayudar a confeccionar el algoritmo.

EJEMPLO 3.2

El algoritmo definido en el Ejemplo 3.1 necesita (N + 1)/2 iteraciones para determinar si el número N es par. El tiempo que tarda el algoritmo en resolver el problema aumentará linealmente con la magnitud del número.

El problema planteado se puede resolver de forma más simple, observando que los números pares son iguales al doble de la parte entera de su mitad. El siguiente algoritmo se basa en esta idea.

Leer *N*

M = 2 * *int(N/2)* *(*int es la función parte entera*)*

Si *M* = *N* **entonces** *escribe "es par"*

si_no escribe *"es impar"*

Este algoritmo resuelve el problema en un tiempo constante, es decir, el tiempo de resolución no depende de la magnitud del número introducido. Normalmente el coste de ejecución de un algoritmo no depende en gran medida del valor de los datos de entrada, y sí varía con el número de datos suministrados. Cuando el procesamiento de cada dato de entrada es independiente del resto el tiempo crece de forma lineal. No obstante, en muchos casos esto no es así.

Por tanto es necesario, una vez diseñado un primer algoritmo, realizar una evaluación del mismo. Si se decide que éste no es eficiente, será necesario o bien diseñar uno nuevo, o bien **optimizar** el original. Optimizar un algoritmo consiste en introducir modificaciones en él tendentes a disminuir el tiempo que necesita para resolver el problema, o a reducir los recursos que utiliza. En muchos casos ambas acciones se contraponen, o sea que una disminución del tiempo implica utilizar más variables. La importancia de cada uno de estos factores dependerá de la naturaleza del problema y de los medios de los que se disponga.

Una vez ideado el algoritmo será necesario crear un programa que sea su transcripción ritmo en algún lenguaje de programación. Las etapas a seguir en este proceso se analizarán más adelante.

3.2 Representación de algoritmos

Hay diferentes métodos para representar los algoritmos. Por supuesto un método es **narrar** o enunciar el algoritmo, como en el Ejemplo 3.2. Para facilitar esta descripción es frecuente utilizar un **lenguaje de descripción de algoritmos** o **"pseudocódigo"**. Existen otros procedimientos de representación que utilizan gráficas o diagramas. Entre estos últimos caben destacar los **organigramas** o **diagramas de flujo** y los **diagramas de Chapin** o **diagramas de Nassi-Schneiderman** (abreviadamente **diagramas N-S)**.

Las próximas secciones se dedican a analizar los tres procedimientos de representación mencionados.

3.2.1 Pseudocódigo

No hay reglas fijas para la representación *narrativa* de algoritmos. No obstante, para describir algoritmos está muy extendido el uso de las estructuras de control del lenguaje **Pascal** o de

APL. En este caso, como el objetivo no es escribir un programa para ser ejecutado por un computador, no hay reglas sintácticas estrictas, el interés se centra en la secuencia de instrucciones. Este tipo de descripción se denomina **pseudocódigo**. La utilización de pseudocódigo presenta las ventajas de: ser más compacto que un organigrama, ser más fácil de escribir, y ser más fácil de transcribir a un lenguaje de programación.

EJEMPLO 3.3

Un algoritmo escrito en pseudocódigo presenta una estructura semejante a la de un programa. El algoritmo del Ejemplo 3.1 no está escrito en pseudocódigo, aunque sí en forma narrativa. Dicho algoritmo se puede escribir en pseudocódigo, tal como se muestra a continuación.

leer *N*

mientras *N > 2* **repetir** *N = N – 2*

si *N = 2* **entonces escribe** *"es par"*

si no escribe *"es impar"*

fin

3.2.2 Organigramas

Los organigramas son herramientas gráficas para representar algoritmos. Un organigrama está compuesto por una serie de símbolos unidos por flechas. Los símbolos representan acciones, y las flechas el orden de realización de las acciones. Cada símbolo, por lo tanto, tendrá al menos una flecha que conduzca a él y una flecha que parta de él (véase Figura 3.1).

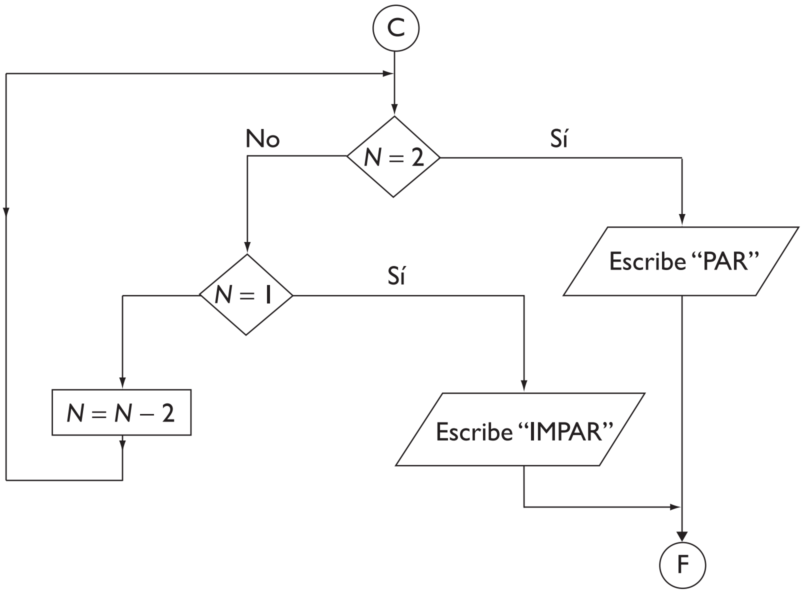


Figura 3.1. *Organigrama del Ejemplo 3.1.*

Cada símbolo usado representa una acción distinta. Hay símbolos específicos para representar las siguientes acciones: **asignación, lectura/escritura, llamada a subrutina, ciclo, decisión, comienzo y fin.** En la Figura 3.2 se muestran los símbolos habitualmente utilizados en la confección de organigramas. En la Sección 3.3 se explicarán con detalle estas acciones.

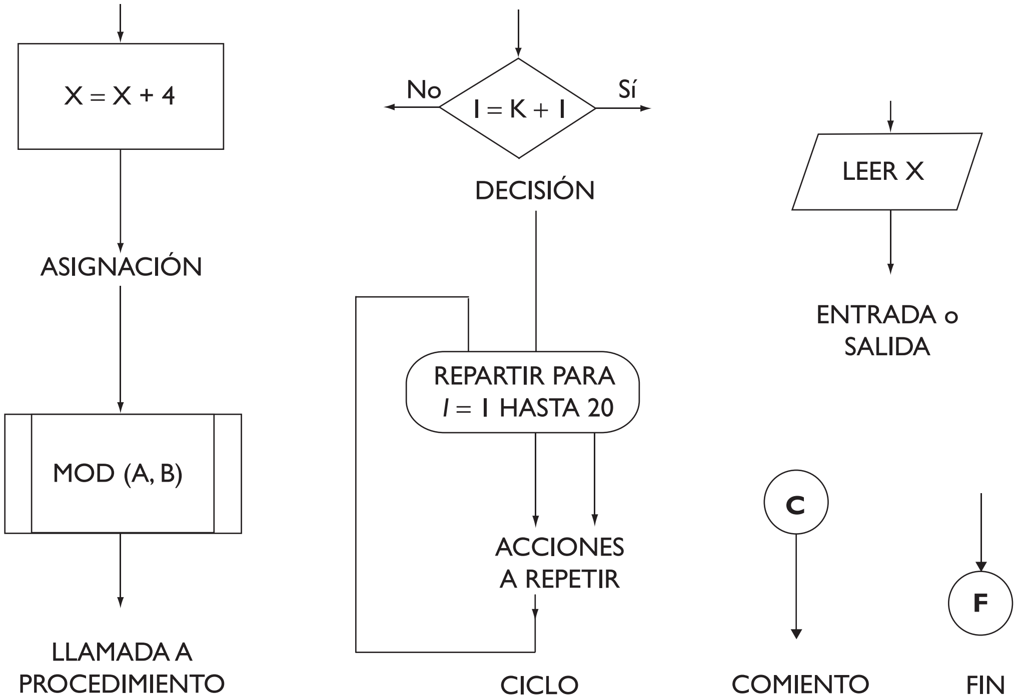


Figura 3.2. *Símbolos usados habitualmente para confeccionar organigramas.*

3.2.3 Diagramas de Nassi-Schneiderman

Los organigramas descritos anteriormente se pueden utilizar para representar cualquier algoritmo. No obstante, los diagramas de Nassi–Schneiderman (o diagramas de Chapin) tienen la ventaja de adecuarse a las técnicas de programación estructurada. Los diagramas N–S no utilizan flechas para indicar el flujo de control. Además estos organigramas tienen otras ventajas respecto a los organigramas clásicos, como son:

- Se leen de arriba a abajo, al igual que un programa estructurado. El organigrama queda por tanto más claro.
- Permiten el uso de técnicas de diseño descendente (véase Sección 3.5.4).
- Favorecen la partición de los programas en módulos pequeños.
- Resaltan más las partes generales, quedando los detalles tanto más pequeños cuanto más específicos.

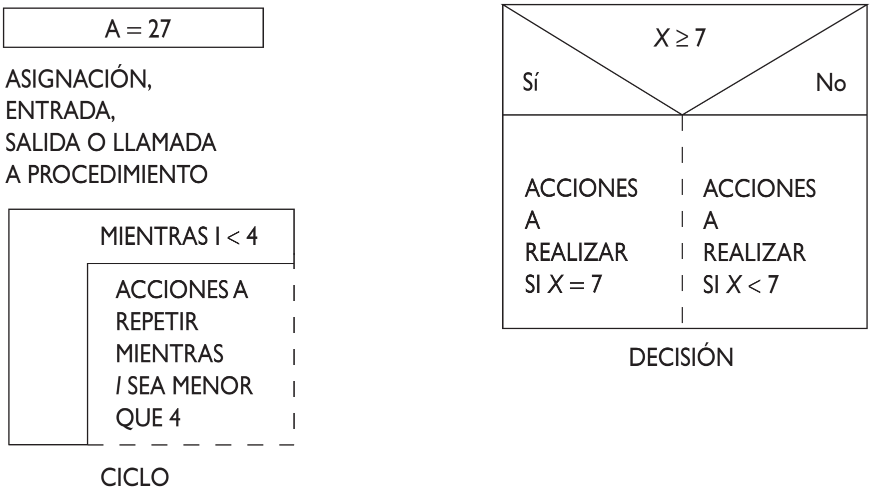


Figura 3.3. *Símbolos usados en diagramas Nassi–Schneiderman.*

Un diagrama N-S es un dibujo contenido en un rectángulo. Dentro del rectángulo se incluyen una serie de símbolos adyacentes. Los símbolos usados corresponden a las formas de transferencia de control: **ciclo, decisión y transferencia secuencial**. Los símbolos usados se muestran en la Figura 3.3.

EJEMPLO 3.4

Los datos relativos a clientes morosos de la empresa Pérez S.A. se almacenan en un fichero secuencial. Los registros de dicho fichero poseen cuatro campos que contienen: Nombre, Fecha de la deuda, Dirección y Cantidad debida. Se desea realizar un programa que efectúe un listado de morosos, como el mostrado en la Figura 3.4. El organigrama de dicho programa puede ser el que se muestra en la Figura 3.5. Nótese como se ha duplicado la lectura, para evitar el uso de una decisión dentro del ciclo.

Pérez, S.A. — Relación de acreedores

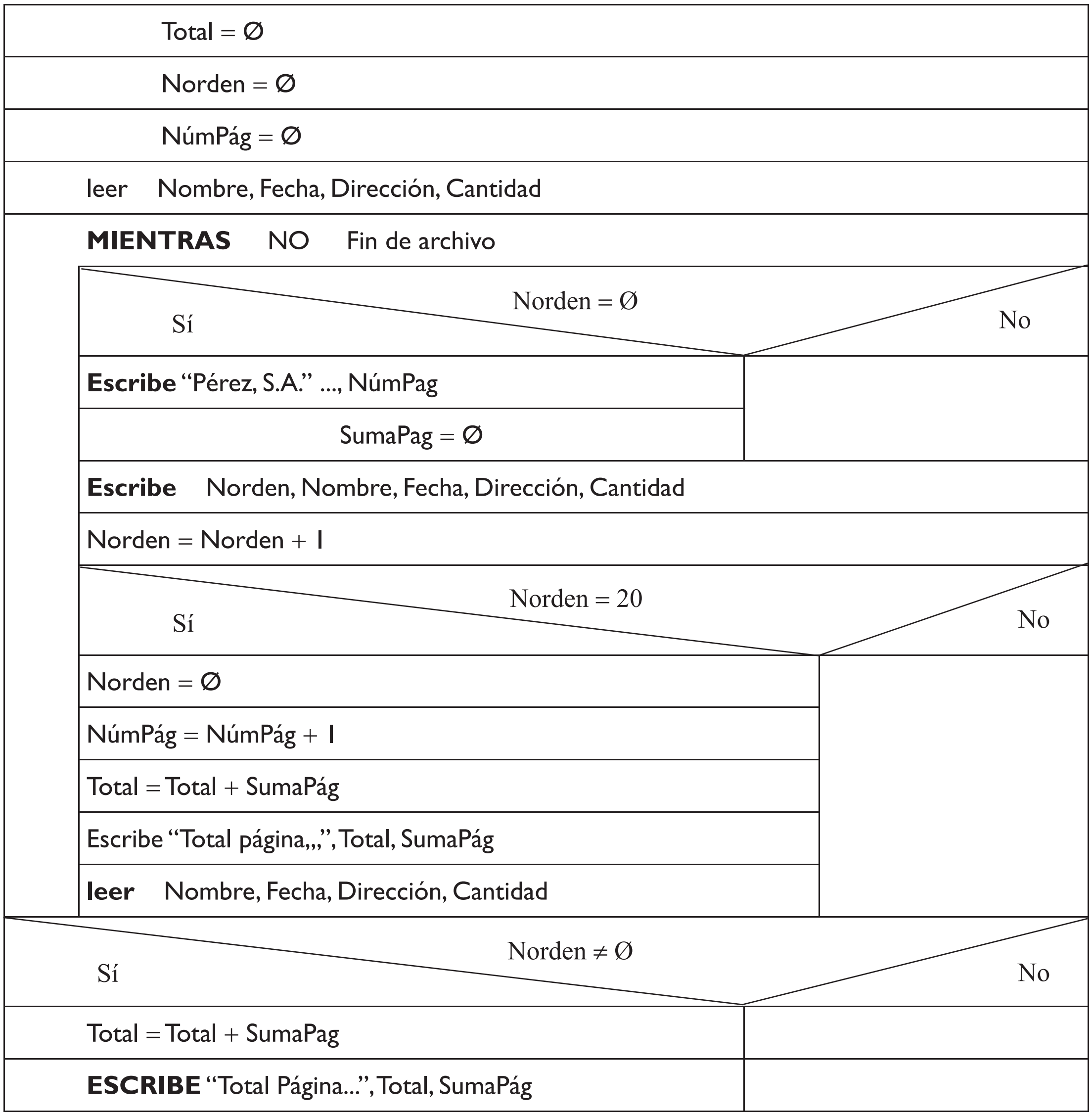
Pág. 1

N. Orden	Nombre	Fecha	Dirección	Cantidad
1	A. Soler	12/12/89	Alamillos, 3	323.50
2	J. López	01/01/90	Prin, 7	1114.50
...	...	...	...	...

Total de página: 2345.73

Total parcial: 2345.73

Figura 3.4. *Listado de acreedores de Pérez S.A.*



Variables:

Total:	Suma de las cantidades
SumaPág:	Suma de las cantidades de una página de listado
Norden:	Número de orden en la página
NúmPág:	Número de página
Nombre:	Nombre de acreedor
Fecha:	Fecha de deuda
Dirección:	Dirección de acreedor
Cantidad:	Cantidad debida

Figura 3.5. *Organigrama del Ejemplo 3.4.*

3.3 Acciones y estructuras de control

A continuación se tratará de las distintas acciones que pueden utilizarse en un algoritmo o en un programa y de las estructuras que se pueden usar para controlar el orden de realización de estas.

Entre las acciones comentaremos las asignaciones y las entradas/salidas. De las estructuras de control se analizarán las decisiones y los ciclos. Se introducirá también el concepto de procedimiento (o subrutina). Éstas no son las únicas acciones y estructuras de control que pueden aparecer en un lenguaje de programación o en la descripción de un algoritmo, pero son suficientes para resolver cualquier problema.

3.3.1 Asignaciones

Una asignación consiste en la evaluación de una expresión y en el almacenamiento de su valor en una variable. Este tipo de sentencias se ejecutan siempre secuencialmente. Esto es, a continuación de la asignación se realiza siempre la acción siguiente del programa. La asignación siempre implica una transferencia (movimiento) de datos en memoria. Esto es un movimiento de datos de unas variables a otras. En algunos casos puede incluir además la realización de una serie de operaciones aritméticas, lógicas, o de manejo de caracteres u otras estructuras de datos (véase Sección 2.1). La variable a la que se le asigna el nuevo valor pierde el que tuviera anteriormente. Esta acción es esencial en la mayoría de los lenguajes de programación (salvo en los lenguajes lógicos y funcionales).

3.3.2 Entradas/salidas

Las operaciones de **entrada/salida** se utilizan para que el programa intercambie información con un medio externo. En una operación de entrada o **lectura**, el programa asigna a una variable un valor almacenado en un soporte masivo o generado por un periférico. En una operación de salida o **escritura** el programa transfiere a un dispositivo de memoria masiva, o a un periférico, el valor de una variable.

En cualquier caso, en la instrucción se debe especificar el dispositivo en el que se escribe, o del que se lee. En algunos casos se omite el nombre del dispositivo, sobrentendiéndose entonces que éste es el terminal o equipo periférico con el que se está accediendo al computador.

Al igual que en las asignaciones, la entrada/salida no conlleva alteración en el orden de ejecución de las acciones siguientes a ella, salvo situaciones excepcionales como la que se indica en el párrafo siguiente.

Si la lectura se realiza de un archivo, se puede llegar a intentar leer más allá del final de éste. En esta situación se dice que se ha producido un error por **fin de archivo**. En un lenguaje de programación esta situación se puede tratar de dos modos. En algunos lenguajes la sentencia de lectura tiene una salida especial para fin de archivo; es decir, se produce un salto en la ejecución del programa si se encuentra un fin de archivo. La otra opción (usada por ejemplo en Pascal) es que con cada archivo del que se lee esté asociado un valor que indica si se ha alcanzado su final. Este valor se actualiza automáticamente después de cada operación. Esto permite que en el programa se puedan tomar distintas acciones según se haya llegado o no al fin del archivo, simplemente comprobando este valor.

3.3.3 Decisiones

Las **decisiones** son acciones de control de flujo. Permiten modificar el orden en que se realizan otras acciones. Una decisión posee siempre un argumento, que normalmente debe ser una

expresión lógica. Dependiendo del valor de la expresión lógica se ejecutan las acciones que están en uno u otro camino, a partir de la decisión. Una decisión permite, por tanto, bifurcar en dos caminos el flujo de acciones.

3.3.4 Ciclos

Un ciclo es una estructura de control que indica la repetición de un segmento de programa. El ciclo puede ser **repetitivo**, en cuyo caso el segmento se repite un número fijo de veces. En este tipo de ciclos existe una variable de control del ciclo a la que automáticamente se le asignan valores sucesivos durante la ejecución del ciclo.

EJEMPLO 3.5

La sucesión de números de Fibonacci se define por la expresión recursiva:

F (k) = F (k – 1) + F (k – 2)

siendo F(0) = 0 y F(1) = 1. Para calcular los N primeros números de Fibonacci habrá que realizar un proceso iterativo tal como se muestra a continuación

Fanterior = 0
Fúltimo = 1
leer *N*
Escribir *Fanterior*
Escribir *Fúltimo*
Para *I = 2* **hasta** *N* **repetir**
Factual = Fanterior + Fúltimo
Escribir *Factual*
Fanterior = Fúltimo
Fúltimo = Factual
fin

***Variables usadas** (todas de tipo entero):*

N:	*Número de términos que se desean generar*
Factual:	*Número de Fibonacci I*
I:	*Contador del ciclo*
Fúltimo:	*Número de Fibonacci I-1*
Fanterior:	*Número de Fibonacci I-2*

En un ciclo repetitivo se debe especificar siempre el valor inicial y final que toma el contador del ciclo y el incremento a aplicar en cada iteración, si este es distinto a uno.

Otra estructura de control usual es el **ciclo condicional**, en el que un segmento de programa se repite **mientras** (ciclo *while*) se cumpla una condición o **hasta que** (ciclo *until*) se deje de cumplir una condición. La condición siempre es una expresión lógica. Este tipo de ciclos, como se verá más adelante es más general que el anterior. En la Figura 3.6 se muestra la lógica de distintos tipos de ciclos.

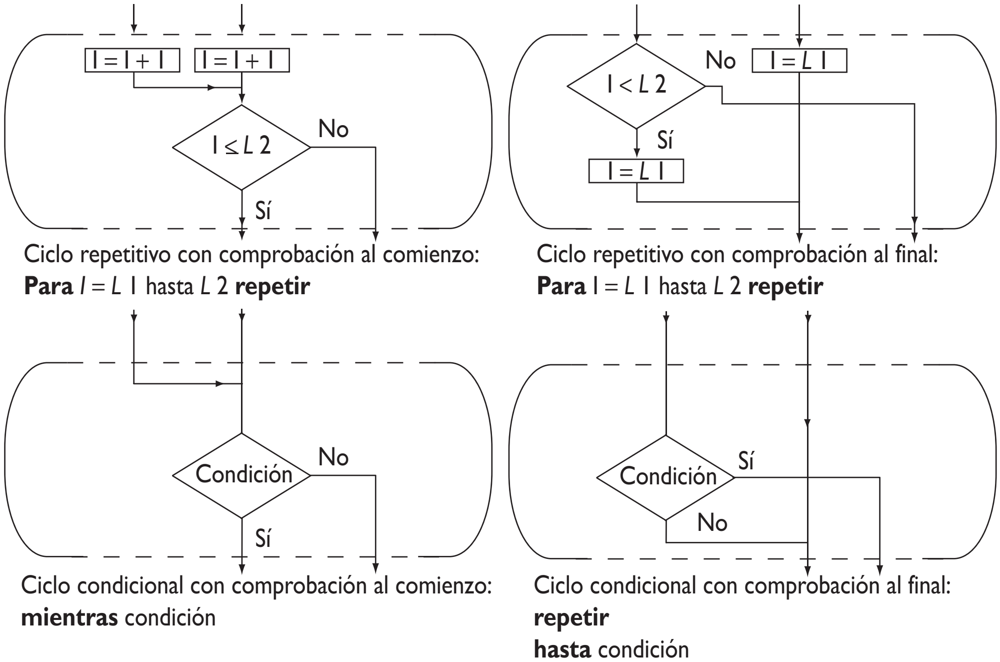


Figura 3.6. *Estructuras de distintos tipos de ciclos.*

EJEMPLO 3.6

Se desea obtener un dígito a partir de un número cualquiera, sumando los dígitos del número original sucesivas veces, hasta que el resultado sea menor que 10. Así, si el dato de partida es 23 el resultado será 2 + 3 = 5, y si es 39 será 3 (ya que 3 + 9 = 12, y 1 + 2 = 3). Para realizar este proceso se puede usar el algoritmo siguiente:

Leer *N*
Suma = N
Mientras *Suma > 9* **repetir**
Resto = Suma
Suma = 0
Mientras *Resto > 0* **repetir**
I = 10[(Resto/10.0)-Resto/10]*
Suma = Suma + I
Resto = Resto/10
Escribe *Suma*
Fin

Variables usadas (Todas de tipo entero):
N: *Número a convertir*
Suma: *Suma de los dígitos*
I: *Dígito*
Resto: *Variable para almacenar suma intermedia*

El segundo ciclo realiza la descomposición del número en dígitos (almacenados en I) y la suma de éstos (en Suma). La descomposición en dígitos se realiza extrayendo dígitos por la derecha. Obsérvese que la primera división (Resto/10.0) es una división real, mientras que la segunda (Resto/10) es entera. La diferencia entre ambas es el primer dígito de Resto (por la derecha) dividido por 10.

3.3.5 Procedimientos

Un **procedimiento, subrutina** o **subprograma** *es un fragmento del programa que realiza una tarea concreta y recibe un nombre por el que puede ser llamado o activado desde otra parte del programa*. Un procedimiento puede tener una serie de variables de comunicación denominadas argumentos, que permiten el paso de información entre el programa y el procedimiento.

El uso de procedimientos evita la duplicación de grupos de sentencias en diferentes partes del programa. Pero sobre todo facilita la construcción y comprensión de los programas, ya que es más fácil diseñar y poner a punto un segmento de programa pequeño, que realiza una función concreta, que uno grande. Sobre este aspecto se volverá más adelante.

EJEMPLO 3.7

Se desea realizar un programa que procese encuestas cerradas. Cada encuesta es una sucesión de valores enteros entre uno y nueve (máximo número de opciones de cada pregunta). Cada número es la respuesta a una de las preguntas de la encuesta. Habrá tantas sucesiones como personas encuestadas. Las encuestas se han grabado en un archivo secuencial que contiene una encuesta en cada registro.

El programa leerá las hojas de respuestas una a una, en una matriz entera (respuesta). Posteriormente cargará los valores de dicha matriz en una **tabla** *de acumulación. La tabla de acumulación es una matriz rectangular de nueve por cien (máximo número de preguntas). Una vez finalizado este proceso se escribirá la tabla.*

A continuación se muestra un algoritmo para realizar este proceso. El algoritmo aparece descompuesto en cuatro procedimientos. Hay un procedimiento principal que resuelve el problema haciendo uso de los procedimientos "Leer_Encuesta", "Acumular" y "Escribir_Encuesta". Esta misma estructura se podría mantener al construir el programa usando un lenguaje de programación.

leer *NP* *(*Número de preguntas por encuesta*)*
leer *NE* *(*Número de encuestas*)*
Para *I = 1* **hasta** *NE* **repetir** *(*Procesar NE encuestas*)*
Leer_Encuesta (Hoja, NP)
Acumular (Hoja NP, Tabla)
Escribir_Encuesta (Tabla, NP)
fin

Variables usadas:
Enteros: NP número de preguntas
NE número de encuestas
I contador de encuestas
Arrays de enteros:
Hoja índice 1..100 (contiene: una hoja de respuestas)
Tabla índices 1..9 y 1..100 (contiene el nº de veces que aparece cada repuesta a cada pregunta). Inicializada: a cero

EJEMPLO 3.7 (Cont.)

Procedimientos usados:

Leer_Encuesta *(Hoja,NP) (*Lee una* **Hoja** *de* **NP** *preguntas*)*
Argumentos: Hoja, NP (definición como en principal)
Variables enteras: K (contador de respuesta)
Para *K = 1* **hasta** *NP* **repetir** *(*Para las NP preguntas*)*
  leer *Hoja [K] (*leer respuesta K*)*
fin *de Leer_Encuesta*

Acumular *(Hoja, NP, Tabla)*
*(*Acumula los resultados de una* **Hoja** *de* **NP** *preguntas en* **Tabla****)*
Argumentos: Hoja, NP, Tabla (definición como en principal)
Variables enteras: J (contador de respuesta)
Para *J = 1* **hasta** *NP* **repetir** *(*Procesar NP respuestas*)*
  Tabla[Hoja [J],J] = Tabla[Hoja [J],J + 1]
  *(*La respuesta Hoja[J] a la pregunta J ha aparecido de nuevo*)*
fin *de Acumular*

Escribir_Encuesta *(Tabla, NP)*
*(*Escribe la* **Tabla** *de resultados de una encuesta de* **NP** *preguntas y Q posibles respuestas. Q es fijo*)*
Argumentos: Tabla, NP (definición como en principal)
Variables enteras: K, J (contadores de preguntas y respuestas)
Escribe *"Resultados"*
para *K = 1* **hasta** *NP* **repetir** *(*para cada pregunta*)*
  escribe *"Pregunta", K*
  para *J = 1* **hasta** *Q* **repetir** *(*para cada respuesta*)*
  escribe *"Respuesta", J, ":" Tabla[J, K]*
fin *de Escribir_Encuesta*

3.4 Recursividad

La recursividad está basada en la utilización de procedimientos, y en una técnica especial de resolución aplicable a problemas con "estructura recursiva". Supongamos un problema "A". Al estudiar el problema A conseguimos dividirlo en dos partes B y C. Si una de estas partes, por ejemplo C, es formalmente idéntica a A, el problema es recursivo, y puede resolverse de forma recursiva.

En términos del lenguaje de programación esto supone utilizar un procedimiento, para resolver el problema, que se va a llamar a sí mismo, para resolver parte del problema.

En cualquier caso la solución recursiva no es nunca la única posible, siempre es factible utilizar un ciclo, en lugar del planteamiento recursivo. No obstante en algunos casos la solución recursiva es la solución natural del problema, y por tanto la más clara.

Al realizar un procedimiento recursivo se está haciendo, implícitamente, uso de una pila, ya que los datos locales del procedimiento se crean cada vez que éste es llamado. Si al resolver el problema es necesario utilizar pilas, el uso de procedimientos recursivos puede ser conveniente. En caso contrario normalmente es desaconsejable, pues una solución recursiva es más lenta que una solución con ciclo.

EJEMPLO 3.8

Se desea construir un procedimiento que lea una lista de números y los escriba en orden inverso. Para resolver este problema se podrían leer todos los números, invertir su orden y escribirlos. No obstante, se podría pensar en una solución más sencilla si se hace una descomposición recursiva del problema. El problema se podría resolver con el siguiente algoritmo:

Leer *N (*primer número*)*
Leer el resto de los números y escribirlos en orden inverso
Escribir *N (*el primer número se escribe el último*)*

El algoritmo consta de tres acciones, una de las cuales es semejante al problema inicial.

Para que esta solución sea aceptable, debe haber un momento en que ya no sea necesario volver a llamar al procedimiento (pues si no, no sería un algoritmo). En este caso la necesidad de llamar al procedimiento desaparece cuando no quedan más números por leer. El algoritmo quedará por tanto como:

```
INVERTIR (*nombre del procedimiento*)
    Leer N
    Si no fin de Archivo entonces (*hay más números*)
            INVERTIR
Escribir N
Fin de INVERTIR
```

3.5 Proceso de creación de un programa

Una vez explicadas las herramientas que se han de usar, estamos en condiciones de exponer qué se ha de hacer para desarrollar un programa en un lenguaje de alto nivel. La concepción de cualquier programa comienza con el **planteamiento** del problema que éste ha de resolver. Cuando se ha adquirido una idea clara y precisa del problema, se debe escoger una **representación** para los diferentes datos que intervienen en él. Hecho esto se puede pensar en redactar un **algoritmo** que resuelva el problema. Este algoritmo ha de depurarse y optimizarse. El paso siguiente es redactar un programa según el algoritmo encontrado, que habrá de ser igualmente depurado. Si hay algún error en el algoritmo, que no fue detectado en el proceso de depuración de éste, posiblemente sea detectado ahora, y habrá que retroceder varios pasos en el proceso. Por último, se debe optimizar y documentar el programa. A continuación se verán con más detalle cada uno de estos pasos.

3.5.1 Planteamiento del problema

Antes de enfrentarse con la resolución de un problema es necesario conocer perfectamente éste. Esencialmente se trata de conocer los resultados a obtener, los datos de que se dispone y

la relación entre ellos. Esto es, saber qué datos son, cuál es su significado, qué valores pueden tomar y qué relaciones hay entre ellos. Aquí se deben detectar omisiones en los datos de partida o ambigüedades en la especificación de los resultados.

3.5.2 Representación de los datos

La estructura que ha de tener el algoritmo que se desea diseñar depende en gran medida de la representación usada para los datos que intervienen en el problema. Si la representación no es la adecuada, el algoritmo no será bueno. Si el problema a resolver es complejo, será necesario establecer un proceso de ajuste entre las estructuras de datos y el algoritmo, retocando las estructuras de datos después de haber ensayado un algoritmo, hasta llegar a una solución aceptable.

EJEMPLO 3.9

Se desea construir un programa que genere, para cualquier número entre 1 y 3999, su representación usando cifras romanas.

El primer paso a realizar es el planteamiento del problema. Hay un único dato de entrada, que es un número entero positivo entre 1 y 3999. Se debe dar un único resultado que es una cadena de caracteres formada por las letras I, V, X, L, C, D y M. La cadena de salida se forma a partir del entero del siguiente modo:

- *para números entre 1 y 10:*

  1→I, 2→II, 3→III, 4→IV, 5→V, 6→VI, 7→VII, 8→VIII, 9→IX, 10→X

- *para números mayores de 10:*

  Se convierte dígito a dígito comenzando por los más significativos. Cada dígito se convierte del mismo modo que las unidades pero sustituyendo los caracteres:

  - *en las decenas se usa X, L y C en lugar de I, V, y X.*
  - *en las centenas C, D y M en lugar de I, V y X.*
  - *en millares M en lugar de I.*

Una vez planteado el problema se debe buscar una representación para los datos. Claramente el dato de entrada es un entero y el de salida una cadena de caracteres. Además de estos es interesante utilizar estructuras de datos para almacenar la representación de los dígitos. Para esto, se podrían pensar varios sistemas. Se puede llegar a una representación concisa y eficiente teniendo en cuenta las reglas que aparecen en la siguiente transformación, que indica cómo generar la representación de unos números a partir de otros:

$2 = 1 + 1$ (II)
$3 = 1 + 1 + 1$ (III)
$6 = 5 + 1$ (VI)
$7 = 5 + 1 + 1$ (VII)
$8 = 5 + 1 + 1 + 1$ (VIII)

Bastaría pues con almacenar la representación de 1, 4, 5, 9,10. La transformación se podría hacer entonces del siguiente modo (para números entre 1 y 9):

si $N \geq 9$ **entonces**
  escribe *"IX"*
  $N = N - 9$

EJEMPLO 3.9 (Cont.)

si *N > 5* **entonces**
  escribe *"V"*
  N = N – 5
si *N > 4* **entonces**
  escribe *"IV"*
  N = N – 4
mientras *N > 1* **repetir**
  escribe *"I"*
  N = N – 1

Para evitar la secuencia de decisiones se puede guardar la representación de estos números en una matriz, de forma que esta secuencia se pueda transformar en un ciclo. Esta idea se podría extender a decenas y centenas. Se puede usar, por tanto, una matriz para almacenar las cantidades no deducibles (1,4,...) y otra matriz de cadenas de caracteres para almacenar su representación. Estas matrices tendrían los siguientes valores (asignados inicialmente):

NUMEROS = 1, 4, 5, 9, 10, 40, 50, 90, 100, 400, 500, 900, 1000

ROMANOS = "I","IV","V","IX","X","XL","L","XC","C", "CD", "D", "CM","M"

El algoritmo quedaría entonces así:

  *I = 13 (*número de elementos en las matrices anteriores*)*
  leer *N*
  mientras *I > 0* **entonces**
    mientras *NUMEROS(I) < N* **repetir**
    N = N – NUMEROS(I)
    escribe *ROMANOS (I)*
    I = I – 1
  fin

  Datos usados: (Además de NUMEROS y ROMANOS)
  I: (entero) índice en las matrices
  N: (entero) número a convertir

Obsérvese que el resultado se genera directamente sobre la salida, no necesitándose una variable para almacenarlo.

3.5.3 Diseño de un algoritmo

Esta es, junto con la fase anterior, la parte más delicada del desarrollo de un programa. Ambas etapas están además íntimamente relacionadas, tal como se ha explicado antes. Indudablemente la dificultad que se encuentre dependerá de la complejidad del problema, entre otros factores.

A nadie se le ocurriría buscar métodos y herramientas para la resolución de un algoritmo para escribir 45. Cualquiera que conozca un lenguaje de alto nivel resolvería este problema con una sentencia. Sin embargo, hay problemas más complejos. Tal como dijimos anteriormente, la experiencia juega un gran papel a la hora de escribir un algoritmo. No obstante nadie debe pensar que a base de escribir programas que escriban 45, u otro número entero, va a poder escribir programas complejos, sentándose delante de una pantalla, como quien escribe una carta.

Hay un límite de complejidad, por encima del cual es difícil moverse sin ayudas, independientemente de la experiencia que se tenga. La mente humana está acostumbrada a trabajar con objetos que puede imaginarse, y en número reducido. Siempre habrá problemas lo suficientemente complejos como para no poder tenerlos en mente, en conjunto.

Esto, no obstante, nos da una idea de cómo se ha de afrontar un problema complejo. Podemos ir descomponiendo el problema a resolver en problemas cada vez más simples, hasta llegar a un nivel que seamos capaces de resolver directamente. Ésta es en esencia la idea del diseño descendente, que trataremos a continuación.

3.5.4 Diseño descendente

El diseño descendente es una técnica natural de desarrollo de algoritmos, hasta el punto de que empezó a usarse intuitivamente antes de haberse sistematizado su empleo. Diversos autores han abordado su estudio.

El método consiste en comenzar trabajando a nivel abstracto, para ir dividiendo el problema en sus partes naturales. De esta forma el problema a resolver se descompone en otros más simples. A estos últimos se les aplica el mismo procedimiento hasta llegar a problemas suficientemente pequeños como para ser resueltos directamente.

Hay otra forma, tal vez más clara, de entender este proceso. Partimos de un problema a resolver y de un repertorio de acciones posibles. Pues bien, olvidémonos de las acciones que podemos realizar, y supongamos que tenemos un ordenador mágico que es capaz de realizar cualquier acción hipotética. Puestas así las cosas, la resolución de cualquier problema puede ser fácil. Las acciones hipotéticas usadas pueden ser complejas y no tendrán que coincidir con las que se tengan en un ordenador real. Una vez encontrado un algoritmo, que usa acciones hipotéticas, se deben considerar individualmente cada una de estas acciones y plantearlas como nuevos problemas, encontrando algoritmos que los resuelvan, para lo que se puede volver a pensar en términos del ordenador mágico. Este proceso se repite sucesivamente hasta llegar a una solución factible del problema.

No obstante no hay que engañarse, puede haber pasos en el diseño que sean especialmente complicados. Hay que pensar que cada nuevo nivel generado debe estar más detallado que el anterior. Esto es, siempre se debe avanzar, pareciéndose cada vez más las acciones hipotéticas a las reales.

Usando este método se van generando soluciones del problema a distintos niveles de detalle. En cada nivel la solución podrá ser comprobada y depurada, antes de pasar a resolver los módulos que la forman a un nivel más bajo. Esto presenta la ventaja de facilitar la tarea de comprobación del algoritmo. El algoritmo, resuelto de esta forma, se puede representar por un árbol en que cada nodo es un módulo (un problema o acción hipotética). Cada subárbol dependiente de un nodo se utiliza en la solución de ese nodo. En particular, el nodo raíz es el problema de partida (*nivel 0*). El algoritmo será correcto si la solución dada a cada nivel es correcta. El proceso descrito para crear un algoritmo se deberá usar para resolver cada nodo del árbol, desde la descripción hasta la depuración. El programa podrá crearse como un todo, teniendo en cuenta al algoritmo completo. Sin embargo, normalmente será más conveniente dividirlo en módulos, coincidentes con las partes naturales del problema encontradas en el momento del diseño. En este último caso, el programa podría construirse de abajo a arriba (de forma ascendente: del nivel n al 0), creando primero procedimientos que resuelvan los módulos de detalle, que una vez comprobados serán usados por otros procedimientos más generales. Hasta llegar a la creación del programa, cuya estructura será la de la solución a nivel 0. En la Figura 3.7 puede verse el árbol de diseño descendente para el ejemplo que se da a continuación.

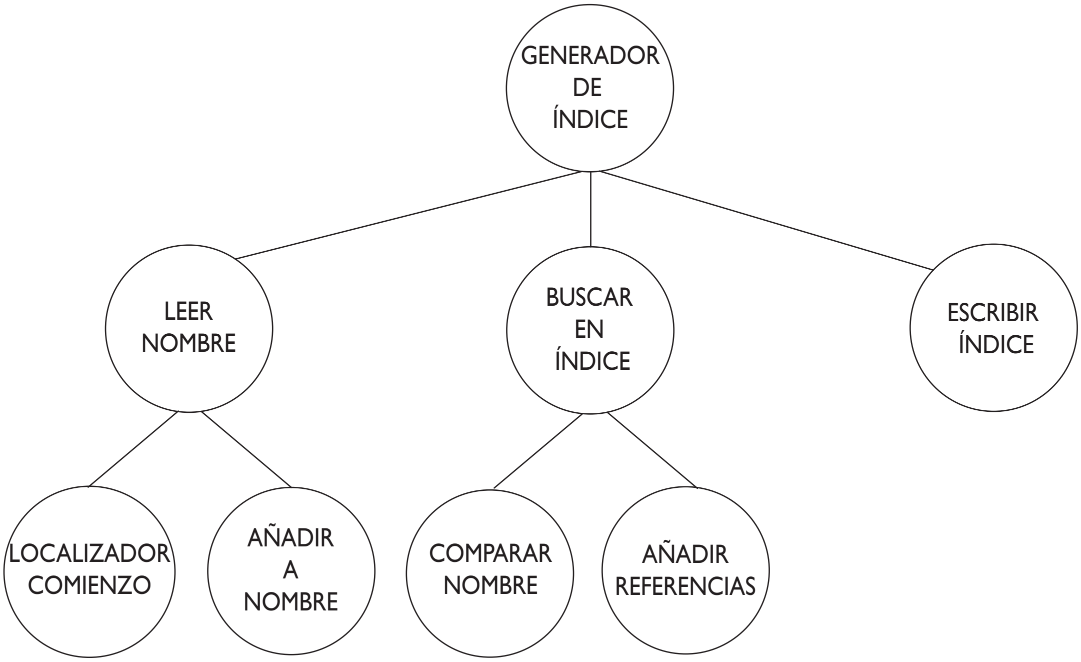


Figura 3.7. *Árbol de diseño descendente correspondiente al Ejemplo 3.10.*

EJEMPLO 3.10

Se desea realizar un programa que construya un índice de un texto. El programa deberá procesar dos archivos. Uno de ellos contiene el texto y el otro una lista de palabras que deben aparecer en el índice. Cada línea de texto está separada de la siguiente por un delimitador (ASCII 13, por ejemplo).

El programa debe generar un listado, en el que aparecerán todas las palabras de la lista, seguidas de los números de línea en que aparecen. Se considera una palabra como una secuencia de letras, dígitos y el carácter "-".

Una solución inicial del problema puede ser:

repetir *Leer_nombre (Nombre, NúmeroLínea)*
Buscar_en_índice (Nombre, NúmeroLínea)
hasta *fin de texto*
Escribir_índice
fin

Con esta solución se está trabajando a un nivel de abstracción alto. El algoritmo anterior, constituye una solución a nivel 0, que será necesario refinar.

Para ello se deberán concretar las acciones abstractas (o hipotéticas) usadas, lo que obliga a concretar las estructuras de datos usadas.

El nombre puede almacenarse en una cadena de caracteres. La longitud máxima de esta cadena limitará el tamaño máximo de la palabra que podrá manejar el programa. Leer un nombre consistirá en leer caracteres, acumunlándolos en la cadena. Al leer se deberán detectar los delimitadores, para poder contar las líneas de texto.

EJEMPLO 3.10 (Cont.)

Leer_nombre *podrá ser por tanto:*
Localiza_principio_de_palabra
Repetir **Leer** *carácter*
Añadir_a_nombre
Hasta *Final_de_palabra*
Fin

Este algoritmo se podría refinar a un nivel más, quedando:

Repetir *Leer_carácter (c)*
Hasta *c < > “ “*
*nombre = “ “ (*cadena vacía*)*
Repetir *nombre = nombre + c (*añadir c a nombre*)*
Leer_carácter (c)
Hasta *c = “ “*
Fin

Se deberá, por último, realizar un algoritmo para **Leer_carácter**. *Este podría ser:*

Leer *c*
Si *c = ASCII(13)* **entonces** *(*delimitador*)*
NL = NL + 1 *(*una línea más*)*
Si *c No_es_letra_permitida* **entonces**
c = “ “
Fin

Para poder refinar la Búsqueda en el índice es imprescindible detallar la estructura del índice. Puesto que las palabras que aparecen en el índice están fijadas en cada ejecución del programa, éste puede ser una estructura de datos estática. Concretamente se podría pensar en un array, en que cada elemento corresponde a una entrada del índice. Los elementos serían de tipo registro, conteniendo cada uno un campo, de tipo cadena, para albergar el nombre correspondiente a la referencia, y un campo de tipo matriz de enteros, para almacenar los números de líneas en que ha aparecido la referencia. El algoritmo para la **búsqueda**, *a nivel 1 podría ser:*

Buscar_en_índice
Comparar_con_nombres_del_índice
Si *Es_igual_a_alguno*
añadir_a_los_números_de_referencia (NL)
Fin

Este algoritmo se refina, quedando a nivel 2 así:

i = 1
No_esta = verdadero
Mientras *No_esta e (i < número de Nombres)*
Si *nombre = Indice[i].nombre* **entonces**
No_esta = falso
i = i + 1
Si *No (No_esta)* **entonces**
i = i – 1
Añadir_a_referencia (i, NL)

Continuando con este proceso se detallaría el algoritmo completo. En la Figura 3.7 se muestra el árbol de diseño descendente del proceso completo.

Otra ventaja del diseño descendente es su adaptación a la programación estructurada. En un programa estructurado la relación que hay entre las distintas sentencias es o bien de contigüidad o bien de inclusión. Un programa estructurado puede, por tanto, representarse en forma de árbol.

Los diagramas N–S pueden jugar también un gran papel, pues se adaptan perfectamente al diseño descendente. La solución a nivel cero se puede representar por un diagrama. Al realizar la solución a nivel uno, se detallarán algunos de los cuadros del diagrama del nivel anterior. Así hasta detallar completamente el organigrama. Si algún módulo interesa desarrollarlo como procedimiento se escribirá su diagrama aparte.

3.5.5 Comprobación y optimización de algoritmos

Un último paso a realizar cuando se escribe un algoritmo es comprobar que es correcto y que realiza las operaciones que se desean. Para comprobar un algoritmo se puede efectuar un seguimiento del mismo. El seguimiento consiste en ejecutar manualmente las operaciones descritas en el algoritmo, según el orden especificado en el mismo, anotando el valor que toma cada dato durante el proceso.

Acción	Datos										
	Hoja		I	J	K	Tabla					
						[1,1]	[1,2]	[2,1]	[2,2]	[3,1]	[3,2]
Para i = 1 **hasta** NE	?	?	1			0	0	0	0	0	0
Leer_Encuesta (Hoja, NP)	?	?									
Para K = 1 **hasta** NP					1						
Leer Hoja (K)	3										
Para K = 1 **hasta** NP					2						
Leer Hoja (K)		2									
Fin Leer_Encuesta	3	2									
Acumular (Hoja, NP, Ta)	3	2				0	0	0	0	0	0
Para J **hasta** NP				1							
Tabla [Hoja [J], J] = ...										1	
Para J **hasta** NP				2							
Tabla [Hoja [J], J] = ...									1		
Fin Acumular	3	2				0	0	0	1	1	0
Para i = 1 **hasta** Ne	3	2	2			0	0	0	1	1	0
Leer_Encuesta (Hoja, NP)	3	2									
Para K = 1 **hasta** NP					1						
Leer Hoja (K)	1										
Para K = 1 **hasta** NP					2						
Leer Hoja (K)		2									
Fin Leer_Encuesta	1	2									
Acumular (Hoja, NP, Ta)		1	2			0	0	0	1	1	0
Para J **hasta** NP				1							
Tabla [Hoja [J], J] = ...						1					
Para J **hasta** NP				2							
Tabla [Hoja [J], J] = ...									2		
Fin Acumular	1	2				1	0	0	0	1	0

Figura 3.8. *Seguimiento del algoritmo del Ejemplo 3.7.*

Si el algoritmo se ha creado por diseño descendente la comprobación se realizará nivel a nivel. En cualquier caso se podrá usar el procedimiento de comprobación descrito anteriormente. En algunos casos se utilizan otros procedimientos de comprobación más rigurosos, que permiten no sólo mostrar que el algoritmo funciona en algunos casos, sino demostrar que algunas partes del mismo son correctas. Concretamente se puede tener en cuenta lo siguiente:

- Para comprobar que un ciclo funciona correctamente se puede aplicar una demostración por inducción matemática.
- Si dos partes del algoritmo son completamente independientes se puede probar el funcionamiento de cada una de ellas independientemente.

En la resolución de cualquier problema es necesario usar variables, para almacenar los datos que se han de manejar. Para que el algoritmo sea legible, se deben especificar en él todas las variables usadas, indicando su tipo y contenido.

EJEMPLO 3.11

Se desea comprobar el algoritmo del Ejemplo 3.7. Para analizar si el algoritmo es correcto se puede efectuar un seguimiento del mismo. Para ello rebajaremos los valores de los parámetros: número de opciones 3, número de preguntas (NP) 2, número de encuestas (NE) 2. Confeccionamos una tabla en la que figuran en columnas las variables del problema, y en filas los distintos pasos de ejecución del mismo. Hecho esto, pasamos a realizar las acciones indicadas en el algoritmo.

Suponiendo la siguiente encuesta de entrada:

3 2

1 2

El seguimiento será el mostrado en la Figura 3.8.

Se comprueba de este modo que la matriz tabla se actualiza correctamente. El seguimiento podría continuar hasta la generación de los resultados.

Obsérvese cómo se ha escrito el comienzo de las acciones del algoritmo a la izquierda, para indicar el punto de la ejecución del algoritmo.

Se ha hecho uso de las líneas horizontales para delimitar los tramos en donde son visibles las variables.

Cuando se produce la llamada a un procedimiento, éste recibe parte de los datos del programa en forma de argumentos, los datos que no se transfieren en esta forma no son utilizables, en principio, por el procedimiento.

La optimización del algoritmo normalmente consiste en buscar uno mejor, para sustituirlo total o parcialmente. Para realizarlo es necesario tener una idea del comportamiento del algoritmo; es decir, realizar una evaluación del mismo. La evaluación es normalmente una tarea compleja, consistente en calcular el número de sentencias que se habrán de ejecutar en la resolución del problema, en el mejor o peor de los casos, y en una situación media. La mayoría de las veces esto no es viable, entre otras cosas por la dificultad para describir un caso de cada tipo sobre papel. Frecuentemente debemos conformarnos con cotas más o menos próximas a los valores reales.

3.6 Conclusión

En este capítulo se ha abordado el estudio los algoritmos, introduciendo distintas representaciones para estos. Los algoritmos se describen usando secuencias de acciones, que se pueden expresar en lenguaje natural o utilizando organigramas. En este capítulo se han descrito las acciones fundamentales en lenguajes de programación procedimentales

Un programa es la transcripción de un algoritmo a un lenguaje de programación, el primer paso en el diseño de un programa es la selección de las estructuras de datos que se deben usar (véase Capítulo 2) y el diseño del algoritmo que se va a seguir para resolver el problema. Para diseñar algoritmos complejos se puede seguir un proceso de diseño descendente.

En este capítulo se ha hecho una descripción general de los conceptos, sin la pretensión de hacer un manual de programación. A los lectores que necesiten profundizar en este tema se les recomienda el estudio, a partir de este punto, de un buen libro de programación (como [Joy03], [Tre89] o [Wir87]).

EJERCICIOS

3.1 Escribir un procedimiento para descomponer el texto contenido en una cadena de caracteres en las palabras que lo componen.

3.2 Desarrollar un algoritmo para calcular el factorial de un número. ¿Sería razonable utilizar una solución recursiva?

3.3 Desarrollar un algoritmo que calcule el día de la semana correspondiente a una fecha dada en la forma día/mes/año.

3.4 Desarrollar un algoritmo que, a partir de una cantidad C, en euros, encuentre todas las descomposiciones posibles en monedas de 1 y 2 euros y billetes de 5, 10, 20, 500 y 100 euros.

Modifica el algoritmo para que trabaje con cantidades reales con dos decimales, y descomponga usando además monedas de 1, 2, 5, 10 20 y 50 céntimos.

Encontrar la descomposición con mínimo número de monedas

3.5 ¿Cómo se pueden representar cantidades enteras de hasta 400 dígitos? Realizar un algoritmo que sume números enteros de hasta 400 dígitos.

3.6 Realizar un algoritmo para encontrar una salida de un laberinto, si ésta existe.

Sugerencias: Considerar el laberinto representado por un array de enteros, en el que los elementos que corresponden a pasillo tengan valor 0 y los que correspondan a muro tengan de valor 2. Marcar las celdas ya visitadas para detectar las situaciones en que no haya salida.

3.7 Hacer un seguimiento del algoritmo del Ejemplo 3.9.

3.8 Redactar un algoritmo para descomponer un número entero en factores primos.

3.9 Redactar un algoritmo para evaluar una expresión aritmética representada como una secuencia de números, operadores y paréntesis haciendo uso de una pila para evaluar los resultados parciales de cada par de paréntesis.

3.10 Un polinomio se puede representar mediante un array de reales en el que cada elemento almacena el coeficiente de un término. Redacta un algoritmo para calcular la derivada de un polinomio representado de este modo.

BIBLIOGRAFÍA

[Goo77] Goodman, S. E.; Hedetniemi, S. T.: *Introduction to the desing and analysis of algorithms*. McGraw-Hill, Kogakusha, 1977.

[Joy03] Joyanes, L.: *Fundamentos de programación*. 3ª Ed. McGraw-Hill, 2003.

[Joy03a] Joyanes L., Rodríguez L., Fernández M.: *Fundamentos de programación. Libro de problemas. Algoritmos, Estructuras de datos.* 2ª Ed. McGraw-Hill, 2003.

[Lew85] Lewis, T. G.; Smith, M. Z.: *Estructuras de datos, programación y aplicaciones*. Paraninfo, 1985.

[Mar03] Martí N., Ortega Y., Verdejo J. A.*: Estructuras de datos y algoritmos. Ejercicios y problemas resueltos*. Pearson Prentice Hall, 2003

[Pri05] Prieto, A.; Prieto, B.: *Conceptos de Informática,* Serie Shaum, McGraw-Hill, 2005.

[Tre89] Tremblay, J. P.; Dedourek, J. B; Bunt, R. B.: *Introduction to Computer Science. An Algorithmic Approach. Pascal Edition*. McGraw-Hill, 1989.

[Wir87] Wirth, N.: *Algoritmos y Estructuras de datos*. Prentice Hall, 1987

[Vin84] Virgos, F.: *Técnicas y elementos de programación, diseño de algoritmos*. Gustavo Gili. 1984.

CAPÍTULO 4

Representación de la información en los computadores

En este capítulo se estudian los aspectos relacionados con la representación de la información en el interior de los computadores. Se consideran cuatro tipos de información: textos, sonidos, imágenes y datos numéricos, ya que cada uno de ellos presenta peculiaridades muy distintas. En definitiva, se trata de hacer comprender al lector los procesos que transforman la información externa al computador en patrones de bits fácilmente almacenables y procesables por sus elementos internos.

También se hacen unas consideraciones generales sobre los problemas que presentan los errores producidos en la transmisión o almacenamiento de información, así como las técnicas más básicas para la detección automática de estos errores.

Según se han ido incrementando los campos de aplicación de la informática, han ido aumentando las necesidades de capacidad de los archivos donde almacenar la información, y además, como consecuencia, han crecido sustancialmente los tiempos necesarios para transmitir dichos archivos a través de redes. Este problema se presenta de forma notable en las aplicaciones multimedia, donde es frecuente utilizar combinadamente textos, sonidos, e imágenes, tanto estáticas como dinámicas. Para reducir el tamaño de los archivos se han desarrollado técnicas de compresión de datos que se estudian en la última sección de este capítulo.

En el Apéndice A se incluyen aspectos prácticos de los sistemas de numeración matemáticos, que son relevantes desde el punto de vista de su aplicación en informática. Este análisis es de gran importancia ya que la mayor parte de las transformaciones de información que tienen lugar en el computador se fundamentan en él, y se recomienda a los lectores no familiarizados con aquellas cuestiones, su lectura antes de iniciar este capítulo.

4.1 Introducción

Un computador es una máquina que procesa información. Más concretamente, la ejecución de un programa implica la realización de unos tratamientos, según especifica un conjunto ordenado de instrucciones (el programa), con unos datos. Obviamente, para que el computador ejecute un programa es necesario darle dos tipos de informaciones: las instrucciones que forman dicho programa y los datos con los que debe operar.

Dos de los aspectos más importantes que se presentan en informática relacionados con la información es cómo *representarla* y cómo *materializarla* o *registrarla* físicamente. Los dos problemas están íntimamente relacionados y el primero de ellos es el objetivo fundamental de

este capítulo. El segundo se resuelve con soportes de información (discos magnéticos y ópticos, y cintas magnéticas, etc) que se estudiarán en los Capítulos 10 y 11.

El diseño de un sistema informático resulta más fácil, su realización menos compleja y su funcionamiento muy fiable si se utilizan sólo dos valores o estados posibles para las variables físicas que representan los caracteres en los soportes de información o, en general, en el interior del computador. Estos valores conceptualmente se representan por cero [0] y uno [1], y corresponden a dos niveles de tensión (0 voltios y 3,5 voltios, por ejemplo), dos valores de corriente (0 y 16 mA, por ejemplo), dos estados de magnetización (N y S), y dos situaciones de un emisor de luz (apagado y encendido), etc. En otras palabras, la información es retenida por medio de dos valores de una magnitud física (**bit**) que en abstracto las podemos representar por cero y uno. En definitiva, haciendo abstracción del soporte concreto de la información, ésta se representa o codifica en el interior del computador por medio de patrones de bits.

Se pueden diferenciar claramente cinco tipos de representaciones:

- **Representación de textos.** Normalmente la información se da al computador en la forma usual escrita que utilizan los seres humanos; es decir, con ayuda de un alfabeto o conjunto de símbolos, que denominaremos caracteres. Las transformaciones de caracteres a patrones de bits tradicionalmente se realizan por medio de **códigos de Entrada/Salida (E/S)** y, en principio pueden definirse de forma arbitraria. No obstante, existen códigos de E/S normalizados que son utilizados por diferentes constructores de computadores; estos códigos se analizarán en la Sección 4.2.
- **Representación de sonidos.** Los computadores actuales, además de procesar textos y datos numéricos, también operan con otros tipos de elementos como sonidos. Las señales de sonido, también denominadas **señales de audio**, suelen ser de voz o música, y son captadas por un sensor que transforma las señales de presión originales en señales eléctricas analógicas. Estas últimas, a su vez, son digitalizadas y de esta forma, según se describe en la Sección 4.3, se representan en el interior del computador.
- **Representación de imágenes.** En la Sección 4.4 del presente capítulo describiremos cómo se efectúa la representación de imágenes estáticas. La representación de imágenes dinámicas, tales como las de video, suele realizarse como un conjunto sucesivo de imágenes estáticas utilizando adicionalmente algún procedimiento de compresión de datos (Sección 4.7).
- **Representación de valores numéricos.** Los datos numéricos se suelen introducir en un computador como cualquier otro conjunto de caracteres; es decir, por medio de un código de E/S para textos. Pero esta representación no es adecuada para operar matemáticamente con este tipo de datos; por lo que, si se van a realizar cálculos con ellos, se deben transformar a una representación fundamentada en el sistema de numeración en base 2, sistema que puede considerarse como una codificación en binario más, pero que al ser una representación numérica posicional es muy apta para realizar operaciones aritméticas. Esto será objeto de análisis en la Sección 4.5.
- **Representación de instrucciones.** Las instrucciones de un programa en lenguaje de alto nivel o en lenguaje ensamblador se dan en forma de texto, representándose en consecuencia por medio de un código de E/S, y el traductor correspondiente, ya sea un compilador o un interprete, se encarga de transformarlas en instrucciones máquina, que no son más que patrones de bits determinados por el repertorio de instrucciones del procesador. En el Capítulo 8 se describen diversos formatos de instrucciones máquina.

Hay dos cuestiones directamente relacionadas con la representación de la información en el interior del computador, y que brevemente se trataran en este capítulo:

- **Detección de errores** de grabación o de transmisión en la información codificada en binario. Existen diversas técnicas para facilitar la detección y corrección automáticas de este tipo de errores, algunas de las cuales incluiremos en la Sección 4.6.
- **Compresión de datos.** Las cantidades de información que procesa un computador han ido creciendo a lo largo de los años, teniéndose con frecuencia que almacenar o transmitir archivos de un gran volumen. Puede reducirse considerablemente la capacidad de almacenamiento y los tiempos de transmisión utilizando **algoritmos de compresión de datos** para disminuir redundancias o recodificar la información. La información así comprimida puede almacenarse y transmitirse ocupando menos espacio y tiempo, respectivamente. Cuando, posteriormente, se desee procesar dicha información puede recuperarse en su forma original utilizando el **algoritmo de descompresión de datos** adecuado. En la Sección 4.7 se incluye una introducción a éstas técnicas.

4.2 Representación de textos

Podemos representar cualquier información escrita (texto) por medio de caracteres. Los caracteres que se utilizan en informática suelen agruparse en cinco categorías:

1) **Caracteres alfabéticos**: son las letras mayúsculas y minúsculas del abecedario inglés:

   A, B, C, D, E, ..., x*, Y, Z, a, b, c, d, ...,* x*, y, z*

2) **Caracteres numéricos**: están constituidos por las diez cifras decimales:

   0, 1, 2, 3, 4, 5, 6, 7, 8, 9

3) **Caracteres especiales**: son símbolos ortográficos y matemáticos no incluidos en los grupos anteriores, entre otros los siguientes:

) (, * / ; : + Ñ ñ = ! ? . “ & > # <] Ç [SP

   Con SP representamos el carácter o espacio en blanco, tal como el que separa dos palabras.

4) **Caracteres geométricos y gráficos**: Son símbolos o módulos con los que se pueden representar marcos de cuadros o de tablas, formas geométricas o iconos elementales, como por ejemplo:

   | ¬ ¯ ♠ ♣ ♥ ♦ ▒ ▓ ░ ╒ ║ ╥ ╔ ╕

5) **Caracteres de control**: representan órdenes de control, como el carácter para pasar a la línea siguiente *(NL)*, o para ir al comienzo de una línea *(CR)* o para la sincronización de una transmisión *(SYN)* o para que se emita un pitido en un terminal *(BEL),* etc.

   Los caracteres de control son insertados en los textos por los usuarios o por los programas de control de periféricos o de comunicación. Por ejemplo, cuando en un teclado, al estar creando un texto, pulsamos la tecla de nueva línea, automáticamente se insertan los caracteres de control *CR* y *NL*. Los circuitos controladores de una impresora que recibe un texto interpretan los caracteres de control y generan las señales de control que provocan las acciones requeridas: saltar al comienzo *(CR)* de la línea siguiente *(NL).*

Al introducir un texto en un computador, a través del periférico correspondiente, los caracteres se codifican con un **código de entrada/salida** de forma que a cada carácter se le asocia

una determinada combinación de n bits. Un **código de E/S** sencillamente es una correspondencia entre los conjuntos:

[4.1]

$$\alpha \equiv \{0, 1, 2, ..., 9, A, B, ..., Z, a, b, ..., z, *, +, \%, ...\} \rightarrow \beta$$

donde:

$$\beta \equiv \{0,1\}^n \equiv \{00...00, 00...01, 00...10, 00...11, ..., 11...00, 11...01, 11...10, 11...11\}$$

Los elementos del conjunto α de caracteres, así como su número (cardinal) m, dependen del código de E/S utilizado por el programa de edición del texto y por el periférico que codifique o decodifique dicho texto. Por otra parte, los elementos del conjunto β son todas las combinaciones posibles de n bits.

Estamos suponiendo que utilizamos un número fijo, n, de bits para codificar los m símbolos de α. El número mínimo de bits necesarios para codificar un conjunto de símbolos depende del cardinal de éste conjunto. Así:

- Con 2 bits $(n = 2)$ podemos hacer 4 combinaciones distintas, con lo que podemos codificar hasta 4 símbolos $(m = 4)$ distintos.
- Con 3 bits $(n = 3)$ podemos hacer 8 combinaciones distintas, con los que podemos codificar hasta 8 símbolos $(m = 8)$ distintos.

  ...
- Con n bits podemos codificar $m = 2^n$ símbolos distintos.

En otras palabras, se verifica que para codificar m símbolos distintos necesitamos n bits, siendo n el menor número natural que verifica:

[4.2]

$$2^{n-1} < m \leq 2^n \qquad ó \qquad n \geq \log_2 m = 3{,}32 \cdot \log(m),\ con\ n \in \aleph$$

Obviamente n debe ser un número natural (entero positivo).

EJEMPLO 4.1

Obtener el mínimo número de bits para codificar las 10 cifras decimales {0, 1,..., 9}

Se necesitarán:

[4.3]

$$n' = 3{,}32 \cdot \log 10 = 3{,}32\ bits$$

es decir, 4 bits (con 3 sólo podemos codificar 8 símbolos).

Dos codificaciones posibles son las que se muestran en la Tabla 4.1.

Obsérvese que, por lo menos, se necesitan 4 bits, pero pueden hacerse codificaciones con más bits de los necesarios. Con 4 bits no se utilizan $2^4 - 10 = 6$ combinaciones, y con 5 bits $2^5 - 10 = 22$ combinaciones.

Tabla 4.1. *Dos codificaciones posibles para los 10 símbolos decimales.*

Alfabeto	*Código I*	*Código II*
0	0000	00000
1	1000	10001
2	0100	01001
3	1100	11000
4	0010	00101
5	1010	10100
6	0110	01100
7	1110	11101
8	0001	00011
9	1001	10010

EJEMPLO 4.2

Uno de los códigos usuales, ASCII, se utiliza para codificar 95 caracteres (es decir, m = 95), por lo que el número de bits mínimo para codificarlos es 7, ya que

$$2^6 < 95 < 2^7$$

Tabla 4.2. *Caracteres alfanuméricos y especiales del código de entrada/salida BCD de intercambio normalizado.*

		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
00	**0**	SP	1	2	3	4	5	6	7
10	**8**	8	9	0	#	@	‘	>	“
20	**16**	:	/	S	T	U	V	W	x
30	**24**	Y	Z	&	.	%	_	>	?
40	**32**	-	J	K	L	M	N	O	P
50	**40**	Q	R	!	$	*	)	;	¬
60	**48**	}	A	B	C	D	E	F	G
70	**56**	H	I	¢	,	<	(	+	|

Como hemos hecho en el Ejemplo 4.1 para los 10 símbolos decimales (Tabla 4.1), podríamos establecer códigos de E/S de forma totalmente arbitraria y con un número de bits *n* dado por [4.2]. Obviamente existen códigos normalizados y que suelen ser utilizados por los constructores de computadores. Hasta mediados de la década de los 70 se utilizaban códigos de 6 bits, como

el **código BCD de intercambio normalizado** (en inglés: *Standard Binary Coded Decimal Interchange Code*) y el **FIELDATA** (utilizado por Sperry-Univac). Estos códigos tienen la ventaja de que requieren poca memoria (6 bits por carácter) pero el gran inconveniente de que con ellos se puede codificar un número muy limitado de caracteres (2^6 = 64 símbolos). Concretamente no se incluyen en el código las letras minúsculas. Una característica de los listados de computador de aquella época es que todos los textos estaban en mayúsculas. En la Tabla 4.2 se incluye el código BCD de intercambio normalizado. La codificación de un texto en un código es muy sencilla ya que consiste en cambiar cada carácter por su código correspondiente, sin más, como se pone de manifiesto en el siguiente ejemplo (*SP* representa el espacio en blanco, que separa dos palabras).

EJEMPLO 4.3

Suponiendo que un teclado utilizase el código BCD de intercambio normalizado qué bits se generarían al pulsar los siguientes caracteres:

754.32 BEATRIZ

La secuencia de bits que se generaría (véase Tabla 4.2) es la siguiente:

754.32 BEATRIZ = 07 05 04 33 03 02 00 62 65 61 23 51 71 31$)_8$ =
000 111 000 0101 000 100 011 011 000 011 000 010 000 000 110 010 110 101 110 001 010 0011 101 001 111 001 011 001$)_{SBCDIC}$

Los códigos más utilizados en la actualidad son el EBCDIC, ASCII, y UNICODE, por lo que a continuación pasamos a describirlos brevemente.

4.2.1 Código EBCDIC

El **código EBCDIC** (*Extended Binary Coded Decimal Interchange Code*) utiliza $n = 8$ bits para representar cada carácter, con el significado que se indica en la Figura 4.1. En la Tabla 4.3 se incluye el código correspondiente a cada carácter. Sumando los valores de la primera fila y primera columna correspondientes a un carácter dado se obtiene el código en hexadecimal de dicho carácter, y sumando los de la segunda fila y segunda columna el código en decimal. El significado de los caracteres de control se incluye en el Apéndice B.

Bits de zona		**Bits de posición**				
0	**1**	**2**	**3**	**4**	**5**	**6**
00 Caracteres sin usar		*00 caracteres A-I*				
01 Caracteres especiales		*01 caracteres J-R*				
10 Letras mayúsculas		*10 caracteres S-Z*				
11 Letras minúsculas		*11 caracteres numéricos*				

Figura 4.1. *Significado de los campos en la codificación EBCDIC.*

Esta normalización permite codificar hasta $m = 2^8 = 256$ símbolos distintos, es decir, posibilita que se representen una gran variedad de caracteres: incluye las letras minúsculas y mayor

número de caracteres especiales. También es posible, y se hace con las combinaciones que empiezan por *00*, codificar caracteres de control que, como se ha indicado anteriormente, suministran órdenes o señales de control, por ejemplo, para impresoras, pantallas o para coordinar las transmisiones de información.

Tabla 4.3. *Conjunto de caracteres EBCDIC (el significado de los caracteres de control se incluye en el Apéndice B).*

		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	**0**	NUL	SOH	STx	ETx	PF	HT	LC	DEL			SM M	VT	FF	CR	SO	SI
10	**16**	DLE	DC1	DC2	TM	RES	NL	BS	IL	CAN	EM	CC	CU1	IFS	IGS	IEA	IUS
20	**32**	DS	SOS	FS		BYP	LF	ETB	ESC			SM	CU2		ENQ	ACK	BEL
30	**48**			SYN		PN	RS	UC	EOT				CU3	DC4	NAK		SUB
40	**64**			â	Ä	à	á	ã	å	ç	ñ	¢	.	<	(	+	|
50	**80**	&	é	ê	Ë	è	í	î	ï	ì	β	!	$	*	)	;	¬
60	**96**	-	/	Â	Ä	À	Á	Ã	Å	ç	Ñ	¦	,	%	_	>	?
70	**112**	ø	É	Ê	Ë	È	Í	Î	Ï	Ì	`	:	#	@	‘	=	“
80	**128**	Ø	a	b	C	d	e	f	g	H	i	«	»	õ	ý	Þ	±
90	**144**	º	j	k	L	m	n	o	p	Q	r	ª	º	æ	¸	Æ	¤
A0	**160**	µ	~	s	T	u	v	w	x	y	z	¡¡	¿	Ð	[	þ	®
B0	**176**	¬	£	¥	·	©	§	¶	¼	½	¾	Ý	¨	¯	]	´	x
V0	**192**	{	A	B	C	D	E	F	G	H	I	-	ô	ö	ò	ó	õ
D0	**208**	}	J	K	L	M	N	O	P	Q	R	^{1}	û	ü	ù	ú	ý
E0	**224**	\	÷	S	T	U	V	W	x	Y	Z	^{2}	Ô	Ö	Ò	Ó	Õ
F0	**240**	0	1	2	3	4	5	6	7	8	9	^{3}	Û	Ü	Ù	Ú	

4.2.2 Código ASCII

El **código ASCII** (*American Standard Code for Information Interchange)* básico utiliza 7 bits y hoy día es de los más usados. Se puede decir que la mayor parte de las transmisiones de datos entre dispositivos se realizan en esta codificación, y corresponde a la normalización ANSI x3.4-1968 o ISO 646. En la Tabla 4.4 se incluye el código correspondiente a cada carácter. Sumando los valores de la primera fila y primera columna correspondientes a un carácter dado se obtiene el código en hexadecimal de dicho carácter, y sumando los de la segunda fila y segunda columna el código en decimal. El significado de los caracteres de control se incluye en el Apéndice B. Usualmente se incluye un octavo bit para detectar posibles errores de transmisión o grabación (bit de paridad, Sección 4.6).

Tabla 4.4. *Conjunto de caracteres ASCII (ANSI-x3.4, 1968).*

		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
00	**0**	NUL	SOH	STx	ETx	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
10	**16**	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
20	**32**	SP	!	“	#	$	%	&	‘	(	)	*	+	,	-	.	/
30	**48**	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	**64**	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	**80**	P	Q	R	S	T	U	V	W	x	Y	Z	[	\	]	^	_
60	**96**	`	a	b	c	d	e	f	g	h	i	j	k	L	m	n	o
70	**112**	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Existen numerosas versiones ampliadas de éste código que utilizan 8 bits y respetan los códigos normalizados del ASCII básico, aprovechando las combinaciones no usadas para representar símbolos adicionales. Entre ellas se encuentran los **códigos ISO 8859-n**, donde n es un número que identifica el juego de los nuevos caracteres introducidos, dependiendo de los lenguajes (Tabla 4.5). Por ejemplo, la norma, **ISO 8859-1**, también denominada **ISO Latín1** (Tabla 4.6) , se proyectó para América y Europa occidental, e incluye vocales con acentos, tildes, con signos diacríticos (esto es, signos ortográficos que dan a una letra un valor especial, como la diéresis usada en alemán y en español), y otras letras latinas no usadas en los países anglosajones. Corresponde a la **página de códigos de E/S** 819 de los PC. Según puede observarse en la Tabla 4.6, en ISO Latín-1 quedan combinaciones libres, que suelen usarse para codificar otros caracteres obteniéndose así nuevos códigos no normalizados, pero compatibles con el ISO Latín1, como ocurre con la página de códigos 850 de los PC.

Tabla 4.5. *Familia de la normalización ISO 8859, ampliaciones de ASCII.*

Denominación	Estándar	Área geográfica
Latín-1	ISO 8859-1	Oeste y Europa del este
Latín-2	ISO 8859-2	Europa central y del este
Latín-3	ISO 8859-3	Europa sur, maltés y esperanto
Latín-4	ISO 8859-4	Europa norte
Alfabeto latín/cirílico	ISO 8859-5	Lenguajes eslavos
Alfabeto latín/árabe	ISO 8859-6	Lenguajes arábigos
Alfabeto latín/griego	ISO 8859-7	Griego moderno
Alfabeto latín/hebraico	ISO 8859-8	Hebreo y Yiddish
Latín-5	ISO 8859-9	Turco
Latín-6	ISO 8859-10	Nórdico (Sámi, Inuit e islandés)

Tabla 4.5. *Familia de la normalización ISO 8859, ampliaciones de ASCII. (Cont.)*

Denominación	Estándar	Área geográfica
Alfabeto Latín/Thai	ISO 8859-11	Lenguaje Thai
Latín-7	ISO 8859-13	Báltico *Rim*
Latín-8	ISO 8859-14	Céltico
Latín-9 (*alias Latín-0*)	ISO 8859-15	Latín 1 con ligeras modificaciones (símbolo €)

Tabla 4.6. *Conjunto de caracteres ISO 8859-1 (Latín 1).*

		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	NUL	SOH	STx	ETx	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
10	16	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
20	32	SP	¡	“	#	$	%	&	‘	(	)	*	+	,	-	.	/
30	48	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	¿
40	64	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	80	P	Q	R	S	T	U	V	W	x	Y	Z	[	\	]	^	_
60	96	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	112	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL
80	128																
90	144																
A0	160		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯
B0	176	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
V0	192	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D0	208	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	x	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E0	224	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	Ì	í	î	ï
F0	240	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	Ü	ý	þ	ÿ

4.2.3 UNICODE

Los códigos de E/S anteriores, y en particular el ASCII básico que es el más utilizado, presentan varios inconvenientes tales como:

- Los símbolos codificados son muy insuficientes para representar los caracteres especiales que requieren numerosas aplicaciones.
- Los símbolos y códigos añadidos en las versiones ampliadas a 8 bits no están normalizados.

- Están basados en un subconjunto muy limitado de los caracteres latinos (el abecedario inglés), existiendo otras culturas que utilizan otros símbolos distintos para las letras y cifras numéricas.
- Los lenguajes escritos de diversas culturas orientales, como la china, japonesa y coreana, se basan en la utilización de ideogramas o símbolos que representan palabras, frases o ideas completas, siendo, por tanto, inoperantes los códigos de texto convencionales, que sólo codifican letras individuales.

Los problemas anteriores son especialmente relevantes para las aplicaciones de Internet, ya que se pretende que éstas sean lo más universales posible.

Unicode es un código de E/S propuesto por un consorcio de empresas y entidades que trata de hacer posible escribir aplicaciones que sean capaces de procesar texto de muy diversos sistemas de escritura. Está reconocido como estándar **ISO/IEC 10646**, y trata de ofrecer las siguientes propiedades:

1) *Universalidad,* ya que persigue cubrir la mayoría de lenguajes escritos existentes en la actualidad,
2) *Unicidad,* ya que a cada carácter se le asigna exactamente un único código, y
3) *Uniformidad,* ya que todos los símbolos se representan con un número fijo de bits, concretamente 16.

Algunas características de Unicode son:

- Cada carácter Unicode está formado por una cadena de 16 bits, pudiendo por tanto codificarse en total 2^{16} = 65.356 símbolos.
- No se contempla la codificación de caracteres de control
- Incluye caracteres combinados, es decir, caracteres que van asociados arriba o abajo con otro símbolo, como por ejemplo los caracteres ñ, ä, ç, ¼, etc. También la combinación de caracteres está permitida para poder crear nuevos elementos de texto, pero debe realizarse por software.
- Unicode no determina la forma o imagen concreta de cada carácter (el *"font"* o fuente), sino que cada combinación representa un concepto abstracto. Un mismo carácter puede ser escrito de distintas formas, así en arábigo una misma letra tiene 4 formas diferentes dependiendo de si se escribe aisladamente, o al principio, en medio o al final de una palabra, todas estas variantes se codifican con una única combinación. No ocurre lo mismo con las letras mayúsculas y minúsculas de los caracteres latinos que tienen códigos distintos.
- También con la misma idea de evitar duplicidades, caracteres muy parecidos en idiomas distintos, tienen igual posición en el código. Esto ocurre por ejemplo con los ideogramas japoneses, chinos y coreanos; aunque su imagen sea distinta, si su significado es el mismo tienen igual código.

La Tabla 4.7 muestra un esquema de cómo se han asignado los códigos Unicode. Las 65.536 diferentes combinaciones (de H'0000 a H'FFFF) se han dividido en cuatro zonas:

Zona A: Comprendida entre las posiciones H'0000 y H'3FFF. En esta zona se incluyen:

- Los códigos ASCII Latín 1 (ISO 8859-1), siendo el primer byte de la combinación Unicode cero. Las posiciones correspondientes a caracteres de control no se codifican.
- El resto de posiciones hasta la H'1FFF esta ocupada por otros caracteres latinos (hasta 336), caracteres griegos (144), cirílicos (256), armenios (96), hebreos (112), etc.

- De la posición H'2000 a la H'3FFFF se codifican símbolos generales como signos diacríticos (signos ortográficos que dan a una letra un valor especial, 112), signos de puntuación (112), subíndices y superíndices (48), símbolos de monedas (48), símbolos matemáticos (256), formas geométricas (96) y signos tipográficos (192); así como los caracteres fonéticos de los lenguajes chino, japonés y coreano.

Zona I: Comprendida entre las posiciones H'4000 a la H'9FFF contiene más de 20.000 caracteres Han, que corresponden a los ideogramas de los lenguajes chino, japonés y coreano.

Zona O: Comprendida entre las posiciones H'A000 y la H'DFFF es un espacio pendiente de asignar. Constantemente hay peticiones de inclusión de nuevos símbolos como la de los Braille, usados por los ciegos.

Zona R: Comprendida entre las posiciones H'E000 y la H´FFFF, es una zona reservada para que los usuarios definan caracteres, o para poder incluir en ella caracteres definidos en códigos de E/S previos.

Tabla 4.7. *Esquema de asignación de códigos en Unicode.*

Zona	Códigos		Símbolos codificados	Nº de caracteres	
A	0000	0000 a 007F	Latín Básico (00 a 7F), definido en la norma ASCII ANSI-x3.4	128	8 192
		0080 a 00FF	Suplemento Latín-1 (ISO 8859-1)	128	
		0100 a 017F	Ampliación A de Latín	128	
		0180 a 024F	Ampliación B del Latín	80	
		0250 a 02AF	Ampliación del Alfabeto Fonético Internacional (IPA)	96	
		02BF a 02FF	Espaciado de letras modificadoras	65	
		0300 a 036F	Combinación de marcas diacríticas (tilde, acento grave, etc.)	112	
		0370 a 03FF	Griego	144	
		0400 a 04FF	Cirílico	256	
		0530 a 058F	Armenio	96	
		0590 a 05FF	Hebreo	112	
		0600 a 06FF	Árabe	256	
		0700 a 074F	Sirio	80	
			etc.		
	3FFF	2000 a 3FFF	Símbolos generales y caracteres fonéticos chinos, japoneses y coreanos	8 192	
I	4000 9FFF		Ideogramas	24 576	
O	A000 DFFF		Pendiente de asignación	16 384	
R	E000 FFFF		Caracteres locales y propios de los usuarios. Compatibilidad con otros códigos	8 192	

EJEMPLO 4.4

En ASCII la cadena de caracteres:

C/Rúa, 7

*se codificaría así (véase Tabla 4.6):*Obsérvese que para obtener la codificación Unicode nos hemos limitado a añadir 8 bits [0] delante de cada código ASCII de 8 bits.

Símbolo original→	C	/	R	ú	a	,		7
ASCII (Latín1, he×adecimal)→	43	2F	52	F9	61	2C	20	37
ASCII (Latín1, binario)→	0100 0011	0010 1111	0101 0010	1111 1001	0110 0001	0010 1100	0010 000	0011 0111

En Unicode, esa misma cadena se codificaría como:

Símbolo→	C	/	R	ú
Unicode (he×.)→	43	002F	0052	00F9
Unicode (bin.)→	0000 0000 0100 0011	0000 0000 0010 1111	0000 0000 0101 0010	0000 0000 1111 1001

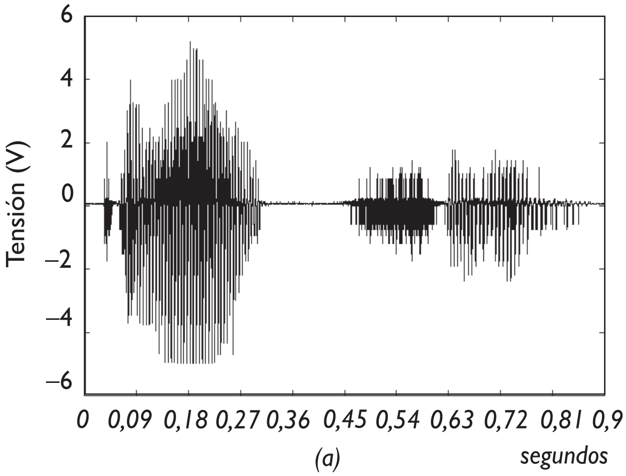
Símbolo *(continuación)*→	a	,		7
Unicode (he×.)→	0061	002C	0020	0037
Unicode (bin.)→	0000 0000 0110 0001	0000 0000 0010 1100	0000 0000 0010 0000	0000 0000 0011 0111

Obsérvese que para obtener la codificación Unicode nos hemos limitado a añadir 8 bits [0] delante de cada código ASCII de 8 bits.

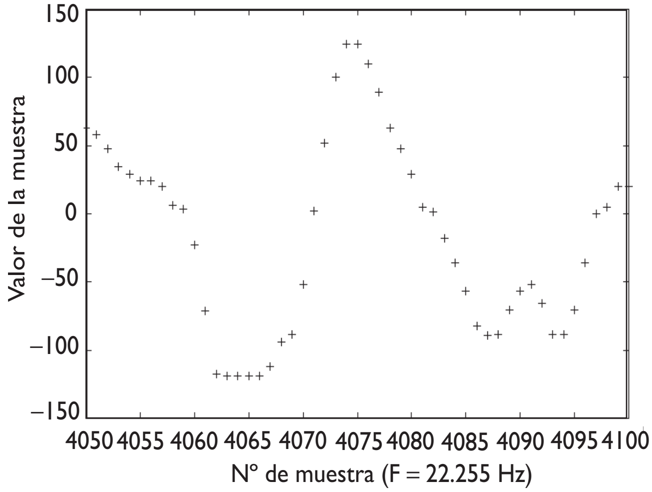
Unicode se está utilizando cada vez más, y el sistema operativo Windows XP, lenguajes como el Java, Perl y C#, y aplicaciones como Netscape y Word lo reconocen. La utilización de Unicode está facilitando la compatibilidad de programas y datos a través de todo el mundo.

4.3 Representación de sonidos

Las aplicaciones multimedia, sobre todo debido al desarrollo de la web, han adquirido una gran importancia. Estas aplicaciones procesan tanto textos como sonidos e imágenes. En esta sección vamos a describir cómo se representan dentro de un computador los sonidos, y en la sección siguiente las imágenes.



(a)



(b)

Posición	***contenido***
1	*2*
2	*2*
3	*2*
4	*2*
.	.
.	.
4050	*63*
4051	*58*
4052	*48*
4053	*35*
4054	*29*
4055	*24*
4056	*24*
4057	*20*
4058	*6*
4059	*3*
4060	*–23*
4061	*–71*
4062	*–118*
-	-
-	-
19996	*1*
19997	*1*
19998	*1*
19999	*1*
20000	*1*

(c)

Figura 4.2. *(a) Señal analógica captada por un micrófono al pronunciar la palabra “casa”; (b) Tramo de muestras comprendido entre 0,184 a 0,186 segundos; (c) valores de las muestras obtenidos por un conversor A/D y que representan a la señal de voz.*

Una señal de sonido se capta por medio de un micrófono que produce una **señal analógica**, esto es, una señal que puede tomar cualquier valor dentro de un determinado intervalo continuo. Posteriormente la señal analógica es amplificada para encajarla dentro de dos valores límites, por ejemplo entre –5 voltios y +5 voltios. En la Figura 4.2a se muestra la señal producida al pronunciar la palabra *casa*. En un intervalo de tiempo continuo (en la figura, entre 0 y 0,9 segundos) se tienen infinitos valores de la señal analógica, por lo que para poder almacenarla y procesarla utilizando técnicas digitales se realiza un proceso de **muestreo**. El muestreo, como su nombre indica, selecciona muestras de la señal analógica a una frecuencia F_s determinada; así cada $T_s = 1/F_s$ segundos se dispone de un valor de la señal. En la Figura 4.2a la señal original se muestrea a una frecuencia de F_s = 22,05 KHz; de esta forma, en el intervalo de tiempo de 0

a 0,9 segundos, se obtiene aproximadamente 20.000 muestras. En la Figura 4.2b se representan las muestras 4.050 a 4.100, que corresponden al intervalo de tiempo que va de 0,184 a 0,186 segundos. Simultáneamente a este proceso, las muestras se digitalizan (transforman a binario) con un **conversor analógico/digital** (Sección 11.5.1.1). En definitiva, la señal de sonido queda representada por una secuencia de valores, por ejemplo de 8 bits, correspondiendo cada uno de ellos a una muestra analógica. En el ejemplo considerado la palabra *casa* quedaría almacenada en un archivo o tabla de 20.000 elementos, conteniendo cada uno de ellos el valor binario de cada una de las muestras correspondientes. En la Figura 4.2c se representan los valores decimales correspondientes al archivo de la señal considerada. A partir de las muestras digitales se puede recuperar la señal.

En el muestreo intervienen dos parámetros fundamentales:

1) la **frecuencia de muestreo**, F_s, que debe ser igual o superior a un determinado valor, que depende de la calidad del sonido a recuperar; en otras palabras, dentro de un intervalo de tiempo dado deben tomarse suficientes muestras para no perder la forma de la señal original, y
2) el *número de bits* (**precisión**) con el que se representa cada muestra, que debe ser el adecuado para poder luego recuperar la señal con la calidad requerida.

Obviamente cuanto mayor es la frecuencia de muestreo y el número de bits por muestra, mayor será el volumen del archivo que almacena el sonido; por lo que ambos parámetros deben elegirse en función de la calidad requerida. No siempre se necesita la misma calidad; así una conversación telefónica no tiene por qué ser de una fidelidad tan buena como la de una audición de música en un equipo de alta fidelidad. En la Tabla 4.8 se especifican parámetros usuales para obtener distintas calidades de sonido.

EJEMPLO 4.5

Obtener la capacidad necesaria para almacenar 1 minuto de una señal de audio estereofónico con calidad CD

Según se indica en la Tabla 4.8, el número de muestras a captar en un minuto es:

[4.4]

$$N = F_s \cdot t = 44100\,\frac{muestras}{segundo} \cdot 60\ segundos = 2{,}646.000\ muestras$$

Como por cada muestra se representa con 16 bits = 2 Bytes, el número de bytes a almacenar por canal será:

[4.5]

$$C_{canal} = 2{,}646.000\ muestras \cdot 2\,\frac{Bytes}{muestra} = 5{,}292.000\ Bytes \approx 5\,\frac{MB}{canal}$$

Pero, al ser una grabación en estereofónico, se necesita el doble de muestras (2 canales):

[4.6]

$$C_{total} \approx 5\,\frac{MB}{canal} \cdot 2\ canales = 10\ MB$$

Tabla 4.8. *Calidades básicas de señales de sonido.*

	Nº de bits/muestra	Frecuencia de muestreo (F_S, KHz)	Periodo de muestreo (T_S, μsegundos)
PCM teléfono	8	8	12,5
Calidad telefónica	8	11,025	90,7
Radio	8	22,05	45,4
CD	16[1]	44,1	22,7

[1] Numero de bits/muestra por canal, al ser en CD el sonido estereofónico hay que multiplicar por 2.

Para procesar la señal analógica producida por un conversor A/D en un dispositivo de entrada, o para generar la señal a aplicar a un conversor D/A en un dispositivo de salida, se utiliza un circuito denominado CODEC, que corresponde a las siglas de CODificador/DECodificador (o COmpresor/DECompresor). Un esquema simplificado de un sistema de grabación y de reproducción de sonido es el que se muestra en la Figura 4.3.

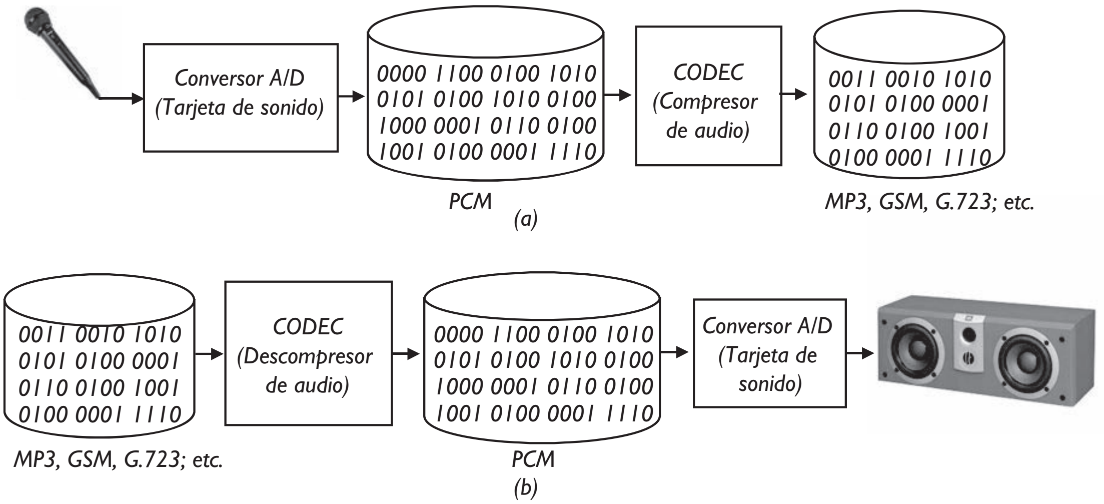


Figura 4.3. *Esquema simplificado de un sistema de (a) grabación digital de sonido, (b) reproducción digital de sonido.*

Hay diversos tipos de CODEC, entre los que se encuentran:

- **PCM** (*Pulse Code Modification*): se transmiten o graba un tren de pulsos correspondientes a cada muestra. El conjunto de pulsos correspondiente a cada muestra se transmite en serie, entre muestra y muestra, tal y como se indica en el ejemplo de la Figura 4.4.
- **DPCM** (*Differential Pulse Code Modulation*): Para disminuir el tamaño de los archivos generados, en vez de almacenar los valores absolutos de cada muestra, para cada muestra se almacena le diferencia con la muestra anterior. Como las variaciones de las señales entre muestras sucesivas son muy lentas, para muchas aplicaciones es suficiente utilizar tan sólo 4 bits para la diferencia (que es el valor que se almacena).
- **ADPCM** (*Adaptive Differential Pulse Modulation*): Dada una secuencia de muestras, un algoritmo predice el valor de la muestra siguiente, y únicamente se almacena el

error entre el valor predicho y el real. De esta forma se necesitan incluso menos bits que en DPCM.

- **u-law**, similar a ADPCM
- **MPEG Audio Capa-III**: para formatos MP2, MP3 y AAC. Como se analizará en la Sección 4.7, las técnicas MPEG se fundamentan en variar el número de bits y la frecuencia de muesreo en función del rango de frecuencias de la señal de audio; teniendo en cuenta que el oído humano es más sensibles a unos rangos de frecuencias que a otros (se guardan más muestras y con mayor número de bits en los rangos de frecuencia para los que el oído es más sensible).
- **CODEC propietarios,** no estandarizados desarrollados por empresas privadas, tales como *Windows Media Audio*, de Microsoft, o *Real Networks,* de Apple, etc.

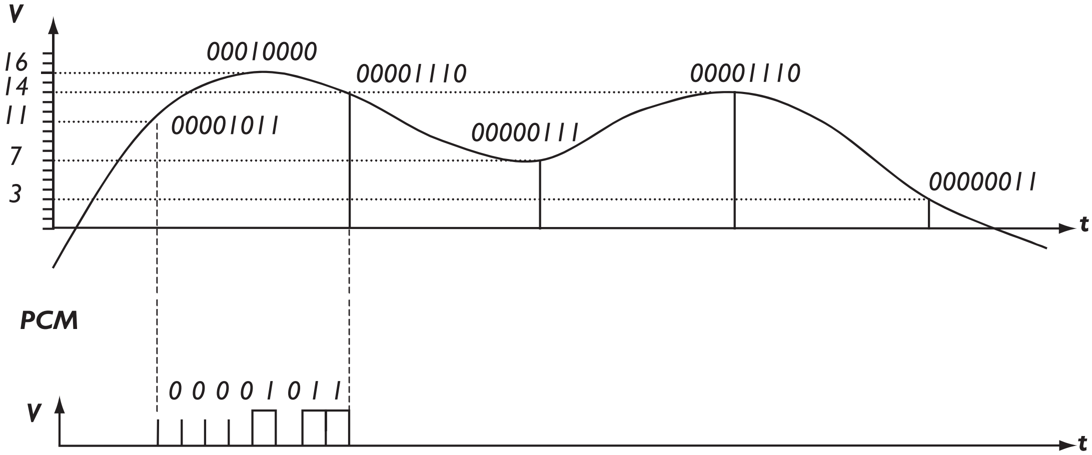


Figura 4.4. *Esquema simplificado de la forma de generación de una información PCM.*

En la Tabla 4.9 se muestran algunos ejemplos de distintos formatos de grabación de audio. Los archivos de audio contienen al principio (antes de los valores de las sucesivas muestras) información adicional sobre las características y formato del archivo.

Tabla 4.9. *Ejemplos de formatos de grabación de señales de audio.*

Formato	Extensión de los archivos	CODEC	Propietario/usuario
AIFF	.aif, .aiff	PCM *u otros*	MAC
AU	.ar	y-law *u otros*	SUN/Next
CD audio 8CDDA)	--	PCM	
MP3	.mp3	MPEG Audio-III	
Windows Media Audio	.wma	*Propietario*	Microsoft
QuickTime	.qt	*Propietario*	Apple Comp.
Real Audio	.ra, .ram	*Propietario*	Real Networks
WAV	.wav	PCM *u otros*	

Se define la **tasa de datos** (*bit rate*, **R**) como *el número de bits que se transfieren por segundo (b/s o bps)*. En un sistema PCM, si F_s es la frecuencia de muestreo y n el número de bits que se deben transferir dentro de un periodo de muestreo, se verificará:

[4.7]

$$R = F_s \cdot n$$

EJEMPLO 4.6

Calcular la tasa de datos de una señal de audio de calidad CD y sin comprimir.

Como cada muestra contiene 16 bits por canal y en cada periodo de muestreo (T_s = 22,7μs) hay que transmitir la información de los dos canales, n = 32 bits; con lo que:

$$R = F_s \cdot n = 44.100 \frac{ciclos}{s} \cdot 32\ bits = 1{,}35\ Mbps$$

4.4 Representación de imágenes

Las imágenes se adquieren por medio de periféricos especializados tales como escáneres, cámaras de video o cámaras fotográficas (Sección 11.4.5). Como todo tipo de información, una imagen se representa por patrones de bits, generados por el periférico correspondiente.

Hay sistemas de codificación de imágenes muy diversos. En la Tabla 4.10 se describen algunos de ellos. Existen dos formas básicas de representar las imágenes, conocidas como mapas de bits y mapas de vectores, respectivamente, y que a continuación se describen brevemente.

Tabla 4.10. *Algunos formatos utilizados para representación de imágenes*.

Tipo	Formato	Origen	Descripción
Mapa de bits	**BMP** *(BitMap)*	Microsoft	Sencillo, imágenes de gran calidad pero ocupa mucho (no se comprimen; inútil para web). Con frecuencia se utiliza para los fondos de pantallas de ordenador y salvapantallas, y en los archivos captados por escáneres de imágenes.
	TIFF *(Tagged Image File Formats)*	Microsoft y Aldus	Prácticamente un estándar para imágenes fotográficas naturales de alta calidad (profundidad de color de 24 bits). Aunque admite compresión LZW, ocupa mucho (no útil para web).
	JPEG *(Joint Photographic Experts Group)*	Grupo JPEG	Calidad razonable para imágenes naturales. Incluye compresión con perdidas, ajustable por el usuario. Usado en la web.
	GIF *(Graphic Interchange Format)*	CompuServe	Muy adecuado para imágenes no naturales (colores planos: logotipos, banderas, dibujos animados, etc.). Utiliza una paleta de hasta 256 colores. No usado para trabajos fotográficos, y muy usado en la web.
	PNG *(Portable Network Graphics)*	Consorcio www	Evolución mejorada de GIF. Muy buena calidad de colores (admite color real). Incluye muy buena compresión.

Tabla 4.10. *Algunos formatos utilizados para representación de imágenes. (Cont.)*

Tipo	Formato	Origen	Descripción
Mapa de vectores	**IGES** *(Initial Ghaphics Exchange Specification)*	ASME/ANSI	Estándar para intercambio de modelos y datos CAD (usable en AutoCAD, etc.)
	DxF *(Document exchange Format)*	ASCII	Formato original del AutoCAD. Estandar de ficheros de texto ASCII para almacenar datos vectoriales para programas CAD.
	PICT *(PICTure)*	Apple Comp.	Imágenes vectoriales que pueden incluir objetos que son imágenes en mapa de bits.
	EPS *(Encapsulated Poscript)*	Adobe Sys.	Ampliación para imágenes del lenguaje de impresión Poscript, con la que se pueden insertar imágenes en distintos formatos como TIFF, WMF, PICT o EPSI.
	TrueType	Apple comp.	Alternativa de Apple y Microsoft para el EPS.

4.4.1 Mapas de bits

Una imagen esta compuesta por *infinitos* puntos, y a cada uno de ellos se le puede asociar un atributo que puede ser su nivel de gris, en el caso de una imagen en blanco y negro, o su color, si la imagen es en color. En consecuencia, para codificar y almacenarse la imagen hay que tener en cuenta dos factores: número de puntos a considerar y código de atributo asociado a cada uno de ellos. Como no podemos almacenar y procesar los atributos de los infinitos puntos, los sistemas de captación consideran la imagen dividida en una fina retícula de celdas, denominadas **elementos de imagen** o **píxeles**, y a cada uno de ellos se le asigna como atributo el nivel de gris o el color medio de la celda correspondiente. En el caso de imágenes en color, éste se suele descomponer en tres colores básicos: rojo (R), verde (G) y azul (B), y la intensidad media de cada uno de ellos en cada celda se codifica por separado. La imagen se representa sencillamente almacenando los atributos de los puntos de la imagen sucesivos en orden, por ejemplo, de izquierda a derecha y de arriba abajo, tal como se muestra en la Figura 4.5. Un archivos de una imagen incluye al principio (o final) información adicional sobre el formato y característica globales del mismo.

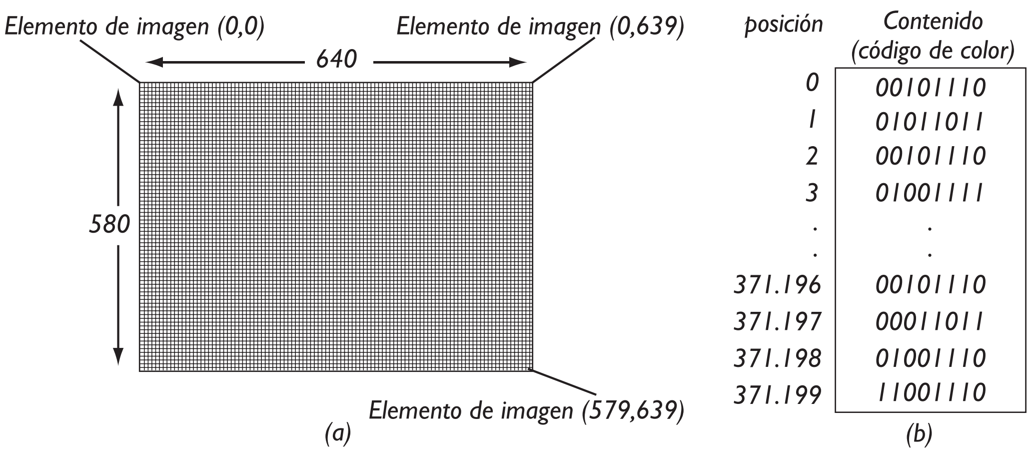


Figura 4.5. *Estructura de una imagen con resolución de 640 × 580 elementos.*

La **resolución de la imagen**, o *(número de píxeles por línea)* × *(número de píxeles por columna)* determina la calidad de imagen (Figura 4.5). La imagen de una fotografía típica también se forma por puntos, y representándola con una resolución del orden de *1280* × *1024* píxeles el ojo humano la considera como continua. En la Tabla 4.11 se indican las resoluciones usualmente utilizadas para digitalización o representación de imágenes (para mas detalles puede consultarse la Sección 11.3.2). Obviamente el tamaño en que se capta o visualiza la imagen influye también en su calidad. Para una misma resolución, cuanto mayor es el tamaño, peor será la calidad.

Tabla 4.11. *Resoluciones usuales para codificar imágenes.*

		Resolución (horizontal × vertical)	**Movimiento**
Convencionales	Fax (A4)	(100, 200, 400) × (200, 300, 400) píxeles/pulgada	Estática
	Foto (8” × 11”)	128, 400, 1200 píxeles/pulgada	Estática
Televisión	Videoconferencia	176 × 144 píxeles/imagen	10 a 36 imágenes/s
	TV	720 × 480 píxeles/imagen	30 imágenes/s
	HDTV (TV alta definición)	1 920 × 1 080 píxeles/imagen	30 imágenes/s
Pantalla computador	VGA	640 × 480 píxeles	
	SVGA	800 × 600 píxeles	
	XGA	1 024 × 768 píxeles	

Además de la resolución, un factor determinante en la representación de un gráfico es el código del **atributo del píxel**. En el caso de imágenes en blanco y negro, se asigna un valor al nivel de gris; así, si se requiriesen 256 niveles de grises, por cada píxel se almacenará un byte ($2^8 = 256$). Para conseguir una gran calidad de colores (calidad fotográfica o color real), la intensidad de cada color básico se codifica con 8 bits; en este caso se necesitarían 3 bytes para codificar cada píxel dando lugar a archivos de una gran capacidad para almacenar una imagen, como se pone de manifiesto en los siguientes ejemplos.

EJEMPLO 4.7

Obtener la capacidad de memoria que ocupará una imagen en blanco y negro con una resolución de 640 × 350 elementos de imagen y con 16 niveles de grises.

Para codificar el atributo (nivel de gris) se necesitan 4 bits, ya que $2^4 = 16$, y en total hay que almacenar el atributo de 640 · 350 = 224.000 elementos. Con lo que la capacidad total será:

[4.8]
$$C = 224.000 \cdot 4 = 896.000\ bits = 109{,}375\ KBytes$$

EJEMPLO 4.8

Obtener la capacidad de memoria que ocupará una imagen en color con una resolución XGA y con 265 niveles para representar cada color básico.

Para codificar el atributo (niveles de los tres colores básicos) se necesitan 3 Bytes, ya que cada color básico necesita 1 byte ($2^8 = 256$).

Por otra parte, de acuerdo con la Tabla 4.11, para obtener calidad XGA hay que almacenar el atributo de 1.024 × 768 = 768 K elementos. Con lo que la capacidad total será:

[4.9]

$$C = 768 \cdot 3 = 2.304\ KB = 2,25\ MBytes$$

Hay formatos que admite que un píxel pueda ser transparente, de forma que al poner debajo de una imagen otra imagen, en dichos píxeles aparecerá esta última. Estos puntos se codifican con un cuarto valor, adicional a los R, G, B, denominado canal alfa (α); que indica si un píxel tiene o no información de la imagen, y por tanto puede hacerse transparente a otra que este en segundo plano.

En la Sección 4.7 se describirán algunas técnicas para compresión de imágenes, que permiten representar las imágenes con un número menor de bytes, con las consiguientes reducciones del espacio de memoria para almacenarla y del tiempo para transmitirla, por ejemplo a través de Internet.

4.4.2 Mapas de vectores

Otros métodos de representar una imagen se fundamentan en descomponerla en una colección de objetos tales como líneas, polígonos y textos con sus respectivos atributos o detalles (grosor, color, etc.) modelables por medio de vectores y ecuaciones matemáticas que determinan tanto su forma como su posición dentro de la imagen. Cuando se visualiza una imagen en una pantalla o impresora determinada, un programa evalúa las ecuaciones y escala los vectores generando la imagen concreta a ver.

Algunas características de este tipo de representación son las siguientes:

- Son adecuadas para gráficos de tipo geométrico y no para imágenes reales (tales como fotografías), ya que los primeros presentan gran cantidad de elementos regulares fácilmente modelables, cosa que no ocurre con los del segundo tipo. En particular resulta muy adecuada en aplicaciones de diseño con ayuda de computador (CAD) en las que se utilizan imágenes compuestas de objetos geométricos que se visualizan y manipulan en dos y tres dimensiones.
- En comparación con la representación con mapas de bits, la representación con mapa de vectores genera usualmente archivos que ocupan menos espacio, y las imágenes son más fáciles de reescalar a cualquier tamaño y de procesar, ya que, por ejemplo, es más rápido mover un objeto o cambiar sus parámetros que recalcular las nuevas posiciones de los miles de elementos de imagen; por el contrario, la calidad y la fidelidad de la imagen en mapa de vectores, en comparación con la realidad, es peor.

En la segunda parte de la Tabla 4.10 se incluyen los formatos más conocidos de representación vectorial de imágenes.

4.5 Representación de datos numéricos

Los datos numéricos se introducen en un computador usando el lenguaje escrito, como secuencias de caracteres, y por tanto se codifican, como cualquier tipo de texto, de acuerdo con un código de E/S, tal como el ASCII. Si se va a realizar algún cálculo matemático, la representación de los datos numéricos como textos es inapropiada. En efecto, como este tipo de codificación no se basa en los sistemas de numeración matemáticos, no podemos aplicar las tablas y reglas de la aritmética para operar con números representados en este sistema (indo-arábigo). Este hecho se pone claramente de manifiesto en el siguiente ejemplo.

EJEMPLO 4.9

Representar en ASCII (8 bits) los números decimales 255 y 42, y obtener el resultado de sumarlos como si estuviesen en el sistema de numeración base 2.

En la segunda columna de la siguiente tabla se tiene la representación ASCII de las cifras de los números correspondientes, que se obtiene sin más que consultar la tabla de códigos ASCII (8 bits) que aparece en la Tabla 4.6. La última fila contiene el resultado de sumar las representaciones binarias como si fuesen números del sistema de numeración base 2. Consultando nuevamente la tabla de códigos ASCII comprobamos que el resultado corresponde a la cadena de caracteres 2ig; muy lejos del resultado buscado que es 297.

Números (caracteres)	Representación en ASCII	Resultado en ASCII
255	0011 0010 0011 0101 0011 0101	
42	0011 0100 0011 0010	
Suma →	0011 0010 0110 1001 0110 0111	→ 2ig

Cuando queramos realizar un cálculo matemático, obviamente lo mejor es representar los datos numéricos en alguna forma que esté basada en el sistema de numeración matemático, como se pone claramente de manifiesto el siguiente ejemplo.

EJEMPLO 4.10

Representar en los números decimales 255 y 42 en binario natural, y obtener el resultado de sumarlos en base 2.

Los resultados se muestran en la siguiente tabla:

Números	Representación en base 2	Resultado en decimal
255	1111 1111	
42	0010 1010	
Suma →	1 0010 0111	→ 297

Del ejemplo se deduce que con la utilización del sistema de numeración base 2 para representar los números se obtiene un resultado correcto, y el número de bits para representar los números es menor que el obtenido con la representación de los mismos como texto, hecho siempre conveniente para ocupar poco espacio en memoria.

La solución adoptada para representar datos numéricos es la siguiente. Cuando se introduce en el computador un número se codifica y almacena como un texto o cadena de caracteres cualquiera. Ahora bien, dentro de un programa, cada dato tiene asociado un **tipo de dato** determinado (véase Capítulo 2). El programador debe asociar a cada dato o variable el tipo adecuado, en consonancia con las operaciones que se realicen con él. Así, por ejemplo, y por lo que respecta a datos numéricos, en el lenguaje C los principales tipos de datos aritméticos son los que se indican en la Tabla 4.12. El programador elige el tipo de dato más adecuado de acuerdo con los objetivos de la variable que define, y teniendo en cuenta que cuanto mayor es el rango y precisión del número, más ocupará la variable en la memoria (ver la columna *Nº de bits* de la tabla).

Tabla 4.12. *Principales tipos de datos aritméticos utilizables en el lenguaje de programación C++ (compilador Borland para PC).*

	Tipo	Nº de bits	Rango de valores	Precisión (dígitos decimales)
Tipos enteros	Carácter	8	–128,127	3
	Carácter sin signo	8	0 a 255	3
	Entero corto	16	–32.768 a 32.767	3
	Entero corto sin signo	16	0 a 65.535	5
	Enumerado	16	–32.768 a 32.767	5
	Entero	*	*	*
	Entero sin signo	*	*	*
	Entero largo	32	–2.147$_1$484.648 a 2.147$_1$ 484.648	10
	Entero largo sin signo	32	0 a 4.294$_1$967.295	10
Tipos reales	Coma flotante	32	± [3,4E – 38 a 3,4E38], 0	7
	Coma flotante doble	64	± [1,7E – 308 a 1,7E308], 0	15
	Coma flotante doble largo	80	± [3,4E – 4932 a 1,1E4932], 0	19

* En máquinas de 16 bits igual a entero corto, y en máquinas de 32 bits a entero largo.

También, en los programas de aplicación existen reglas que determinan si un dato se considera como texto o como número. Así, en la Tabla 4.13 se indican las reglas del programa Excel (de creación de hojas electrónicas) para que un dato sea considerado como número y no como texto.

Tabla 4.13. *Reglas del programa Excel para introducir un dato y sea considerado como número.*

El texto tecleado:	– Deberá comenzar con un digito, o uno de los siguientes caracteres: +, –, =, , – Puede terminar con el símbolo % – Puede tener como máximo una coma decimal – Puede introducirse en *notación científica*; por ejemplo: *2,7E-15*

En la columna 3 de la Tabla 4.12 se observa que cada tipo de datos ocupa un número determinado de bits. En cualquier caso, el dato completo debe encajar en palabras de memoria, de forma que si el tamaño del dato es mayor que la longitud de palabra de memoria, aquel se trocea convenientemente, como indica el ejemplo de la Figura 4.6. Una vez dividido el patrón de bits que representa al dato en porciones que encajen exactamente en palabras de memoria, dichas porciones se almacenan en posiciones consecutivas; pero hay dos posibilidades de almacenamiento:

- Almacenar primero (en las posiciones más bajas de memoria) la parte menos significativa del dato (Figura 4.6b); este convenio se denomina **criterio del extremo menor**, y
- Almacenar primero la parte más significativa del dato (Figura 4.6c); este es el **criterio del extremo mayor**.

Los mismos criterios se utilizan para almacenar cadenas de caracteres (Figura 4.7). Aunque seguir un criterio u otro es irrelevante, no hay un convenio que establezca cual de los dos debe utilizarse, así hay computadores que siguen el primero y otros el segundo[1].

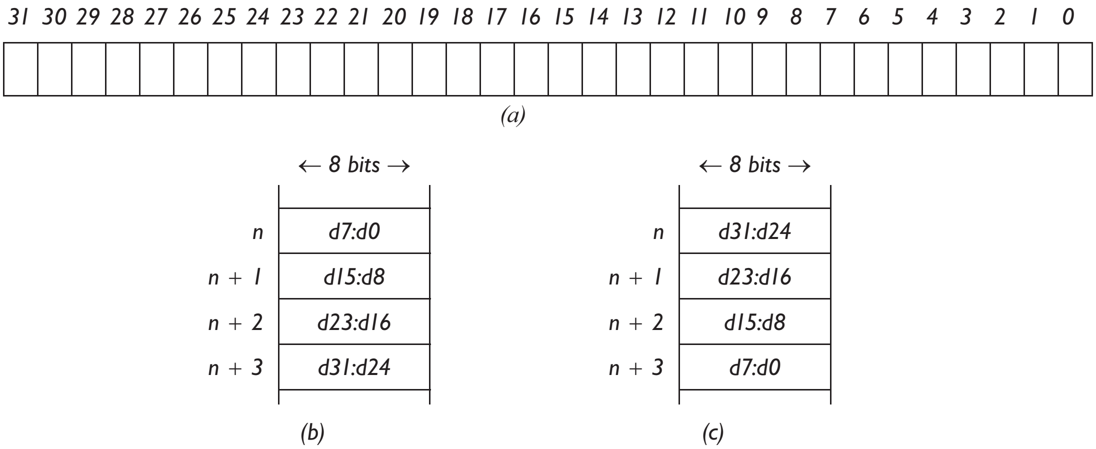


Figura 4.6. *Formas de almacenar un dato numérico (d31:d0) de 32 bits en una memoria direccionable por bytes: (a) dato original; (b) criterio del extremo menor, y (c) criterio del extremo mayor.*

Una vez definidos los datos numéricos de un programa o de una aplicación, una rutina de la biblioteca del lenguaje de programación se encarga de transformar la cadena de caracteres que simboliza el número en la representación numérica.

Hay dos formas básicas de representar los datos numéricos: como números enteros o como números reales. La representación de números reales se suele conocer también como representación en coma flotante o representación científica o representación exponencial. Las Secciones 4.5.1 y 4.5.2 se dedican a describir los principios de estas representaciones, respectivamente. En todos los casos denominaremos N al dato, n al número total de bits dedicados a almacenarlo, y los bits individuales del dato como indica la Figura 4.6a, es decir, el bit más significativo (el de la izquierda) es el $n - 1$ y el menos significativo (el de la derecha) el 0.

[1] En inglés, el criterio del extremo menor se denomina **little-endians**, y el del extremo mayor **big-endians**. Estas denominaciones se basan en una anécdota descrita en la novela “Los viajes de Gulliver” de Jonathan Swift, que pone de manifiesto la irrelevancia que se presenta muchas veces de seguir uno u otro criterio, pero la dificultad en ponerse de acuerdo en uno concreto de ellos.

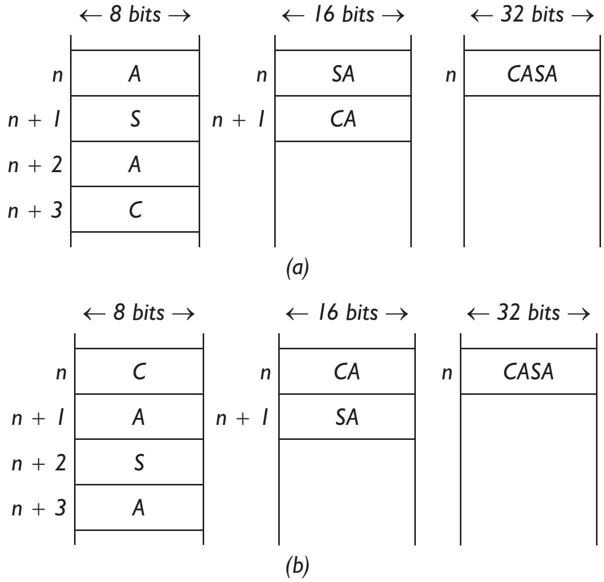


Figura 4.7. *Cadena de caracteres "CASA" almacenada en una memoria de palabras de 8, 16 y 32 bits: (a) criterio del extremo menor, y (b) criterio del extremo mayor.*

4.5.1 Datos de tipo entero

Se distinguen (Tabla 4.14) dos formas básicas de representar en el interior del computador los datos de tipo entero: **representación binaria** y representación de **dígitos decimales codificados en binario** (o **representación BCD**). A su vez, dentro de la representación binaria se tienen dos situaciones, representación sin signo y representación con signo. En este último caso es usual considerar cuatro métodos diferentes: signo y magnitud, complemento a 1, complemento a 2, y representación sesgada (o por exceso). En la representación BCD, a su vez, cada digito decimal se puede codificar de tres formas distintas: natural, Aiken y exceso de 3; aquí únicamente describiremos la natural. Los lenguajes de programación admiten simultáneamente distintas representaciones para los números enteros (un ejemplo está en la Tabla 4.12), y el programador elige para cada variable que define la más adecuada. A continuación se describen las diferentes representaciones citadas.

Tabla 4.14. *Diferentes representaciones de datos enteros.*

Representación binaria	– Enteros sin signo	
	– Enteros con signo	• Signo y magnitud
		• Complemento a 1
		• Complemento a 2
		• Sesgada (o en exceso)
Representación de dígitos decimales codificados en binario (BCD)	– BCD natural	
	– BCD Aiken	
	– BCD exceso en 3	

4.5.1.1 Enteros sin signo

En este caso todos los bits del dato representan el valor del número expresado en binario natural (sistema de numeración base 2) (Figura 4.8a). Así, con $n = 8$ bits se pueden representar los números:

Binario	Decimal
0000 0000	0
0000 0001	1
0000 0010	2
....	...
....	...
1111 1101	253
1111 1110	254
1111 1111	255

En general, como hay *n* bits para representar el número, los valores mayor y menor representables son:

[4.10]

$$N(mín) = 0 \quad ; \quad N(máx) = 2^n$$

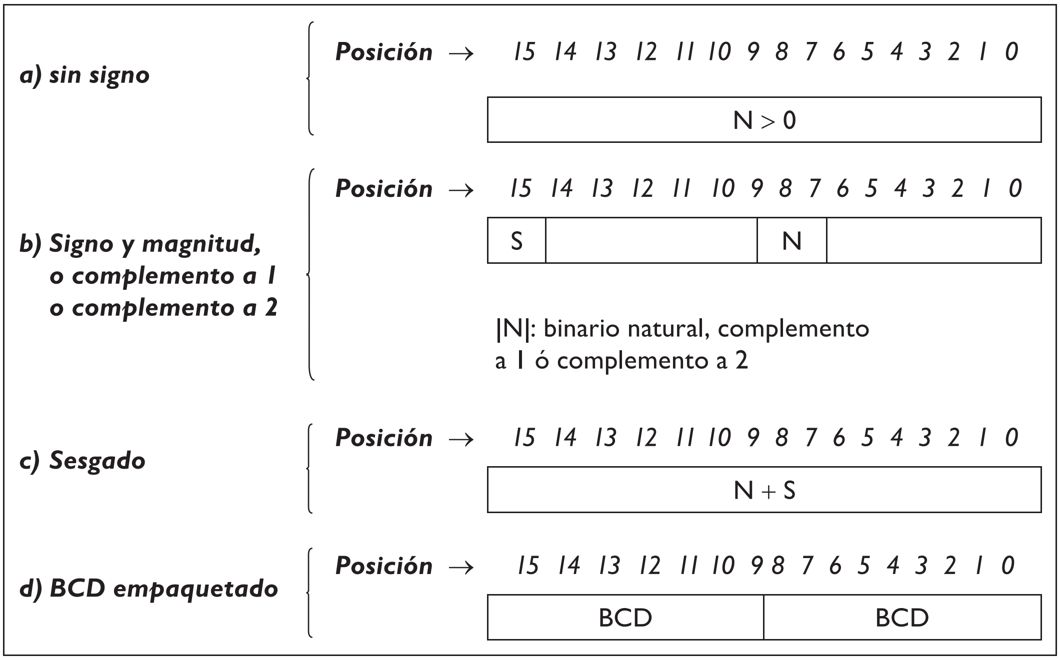


Figura 4.8. *Esquemas de distintas representaciones de datos de tipo entero.*

4.5.1.2 Enteros en signo y magnitud

El signo se representa con el bit más significativo del dato (bit $n-1$). Este bit es 0 si el número es positivo y 1 si el número es negativo. El resto de los bits ($n-2$ a 0) representan el valor absoluto del numero en binario natural (Figura 4.8b).

En general, como hay $n-1$ bits para representar la magnitud del número, los valores mayor y menor representables son:

[4.11]

$$N(mín) = -(2^{n-1}-1) \quad ; \quad N(máx) = 2^{n-1}-1$$

En la segunda columna de la Tabla 4.15 se muestra un ejemplo con todos los números representables con $n = 4$ *bits*. Obsérvese que hay dos representaciones para el cero. Este hecho puede dar lugar a problemas ya que si se desea comprobar si el resultado x de una operación es *0,* habrá que verificar tanto si es igual a $+0$ como a -0, caso de que esta eventualidad no haya sido prevista por el compilador o la unidad aritmético lógica.

4.5.1.3 Enteros en complemento a 1

El signo se representa de la misma forma que en el caso de signo y magnitud. El resto de los bits ($n-2$ a 0) representan, si el número es positivo $(N > 0)$ el valor absoluto del numero en binario natural, y si no $(N < 0)$ su complemento a 1 (Figura 4.8b).

En general, como hay $n-1$ bits para representar la magnitud del dato, los números mayor y menor representables son:

[4.12]

$$N(mín) = -(2^{n-1}-1) \quad ; \quad N(máx) = 2^{n-1}-1$$

En la tercera columna de la Tabla 4.15 se muestra un ejemplo con todos los números representables con $n = 4$ *bits*. Obsérvese que, al igual que ocurre en la representación en signo y magnitud, hay dos representaciones para el cero; teniéndose que realizar las mismas previsiones que en aquel caso.

4.5.1.4 Enteros en complemento a 2

El signo se representa de la misma forma que en el caso de signo y magnitud. El resto de los bits ($n-2$ a 0) representan, si el número es positivo $(N > 0)$ el valor absoluto del numero en binario natural, y si no $(N < 0)$ su complemento a 2 (Figura 4.8b).

En la cuarta columna de la Tabla 4.15 se muestra un ejemplo con todos los números representables con $n = 4$ *bits*. Obsérvese que aquí no hay dos representaciones para el cero, y el número menor es *B'1111*, teniendo una combinación más para representar un número negativo más. En este caso, por tanto, los números mayor y menor representables son:

[4.13]

$$N(mín) = -2^{n-1} \quad ; \quad N(máx) = 2^{n-1}-1$$

4.5.1.5 Representación sesgada

En la representación sesgada, sencillamente se le suma al número N un sesgo S, de forma tal que el número resultante siempre es positivo, no siendo necesario reservar explícitamente un bit de signo (Figura 4.8c). El dato almacenado es el valor de $N + S$ en binario natural. Usualmente se toma como sesgo $S = 2^{n-1}$. Esta notación también se le suele denominar **representación con exceso**.

En la quinta columna de la Tabla 4.15 se muestra un ejemplo con todos los números representables con $n = 4$ *bits*. Aquí el sesgo es $S = 2^3 = 8$. Obsérvese que en este caso los números mayor y menor representables son:

[4.14]

$$N(mín) = -2^{n-1} \quad ; \quad N(máx) = 2^{n-1} - 1$$

También hay que hacer notar que en la representación sesgada los números positivos empiezan por 1 y los negativos por 0.

Tabla 4.15. *Alternativas usuales de representación de datos de tipo entero con signo en el supuesto de datos de n = 4 bits.*

	Tipo de representación			
Nº decimal	**Signo y magnitud**	**Complemento a 1**	**Complemento a 2**	**Sesgado**
7	0111	0111	0111	1111
6	0110	0110	0110	1110
5	0101	0101	0101	1101
4	0100	0100	0100	1100
3	0011	0011	0011	1011
2	0010	0010	0010	1010
1	0001	0001	0001	1001
+0	0000	0000	0000	1000
–0	1000	1111	----	----
–1	1001	1110	1111	0111
–2	1010	1101	1110	0110
–3	1011	1100	1101	0101
–4	1100	1011	1100	0100
–5	1101	1010	1011	0011
–6	1110	1001	1010	0010
–7	1111	1000	1001	0001
–8	(11000)	----	1000	0000

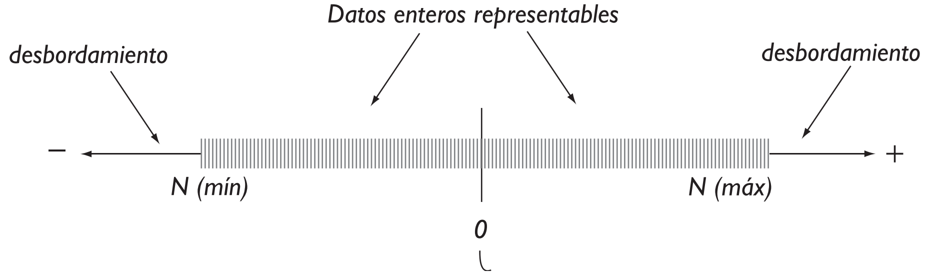


Figura 4.9. *Rango de posibles representaciones de datos enteros.*

En la Tabla 4.16 se incluyen los límites máximo y mínimo de los números enteros para distintas longitudes de palabra, *n*, y diferentes tipos de representación. Si, como resultado de una operación, se tuviese que obtener un número tan grande en valor absoluto que no fuese representable, por ser *n* insuficiente, la ALU genera un patrón de bits que no es valido, y se dice que se ha producido un **desbordamiento** (*overflow*); en este caso el resultado debe desecharse (Figura 4.9).

La forma más utilizada en la actualidad para representar los números enteros con signo es la de complemento a dos. Ello se debe a la facilidad de efectuar las sumas y restas con este tipo de representación (véase Apéndice A, y Sección 9.1.2.1). La representación sesgada, se utiliza entre otras situaciones para representar los exponentes de los datos de tipo real, como se indica en la Sección 4.5.2.

Tabla 4.16. *Valores enteros máximos y mínimos representables con distintos sistemas.*

Representación	Valor mínimo		Valor máximo	
	Patrón binario	Decimal	Patrón binario	Decimal
Sin signo	0000···0000	$N(mín) = 0$	1111···1111	$N(máx) = 2^n - 1$
Signo y magnitud	1 | 1111···1111	$N(mín) = -(2^{n-1} - 1)$	0 | 1111···1111	$N(máx) = 2^{n-1} - 1$
Complemento a 1	1 | 0000···0000	$N(mín) = -(2^{n-1} - 1)$	0 | 0000···0000	$N(máx) = 2^{n-1} - 1$
Complemento a 2	1 | 0000···0000	$N(mín) = -2^{n-1}$	1 | 0000···0000	$N(máx) = 2^{n-1} - 1$
Sesgado	0000···0000	$N(máx) = -2^{n-1}$	1111···1111	$N(máx) = 2^{n-1} - 1$

EJEMPLO 4.11

Obtener la representación interna del número N = –3675 en los cuatro formas vistas anteriormente, suponiendo que se tiene que almacenar en 2 Bytes.

1) *Transformamos a binario el valor absoluto del número, para lo cual previamente lo pasamos a hexadecimal:*

```
3675 |16
047    229 |16
 155   069  14
  11    05
```

Expresando el último cociente y los restos sucesivos en hexadecimal (véase Apéndice A): 14 → E, 5→ 5, 11 → B; es decir,

D’3675 = H’0E5B = B’ 0000 1110 0101 1011

EJEMPLO 4.11 (Cont.)

2) Representación en signo y magnitud:

Signo: como N < 0, será d(n – 1) = 1, con lo que:

1000 1110 0101 1011

o, abreviadamente: ***8E5B***

3) Representación en complemento a 1

Como el número es negativo, hay que obtener el complemento a 1 de N, para lo cual podemos utilizar el valor hexadecimal de N (0E5B) obtenido anteriormente:

```
  FFFF
– 0E5B
  ----
  F1A4 o, en forma ampliada:  1111 0001 1010 0100
```

(Observar que en el resultado se obtiene directamente el signo: d(n – 1) = 1.

4) Representación en Complemento a 2

Añadimos 1 al resultado anterior

```
  F1A4
+ 0001
  ----
  F1A5 o, en forma ampliada:  1111 0001 1010 0101
```

5) Representación sesgada

Tenemos que añadir al valor de N el sesgo, que en este caso será:

$$S = 2^{n-1} = 2^{15} = \text{B'}1000\ 0000\ 0000\ 0000$$

Hagámoslo, por ejemplo en binario (como N < 0, en realidad tenemos que efectuar S – |N|):

```
  1000 0000 0000 0000
– 0000 1110 0101 1011
  -------------------
  0111 0001 1010 0101
```

Es decir, la representación interna del número será:

0111 0001 1010 0101; o abreviadamente: ***71A5***

4.5.1.6 Datos enteros representados con dígitos decimales codificados en binario (BCD)

En ocasiones los datos de tipo entero se representan internamente codificando aisladamente cada dígito decimal con cuatro dígitos binarios, según la Tabla 4.17. De esta forma, en un byte se pueden representar 2 dígitos decimales, denominándose esta representación **BCD empaquetada** (Figura 4.8d), o bien un único digito decimal, obteniéndose una representación **BCD desempaquetada**.

Esta forma de codificar es poco eficiente, puesto que de las $2^4 = 16$ combinaciones posibles de 4 bits sólo se utilizan 10 (Tabla 4.17). No obstante, esta representación a veces se utiliza por la proximidad a nuestro sistema decimal y por la gran facilidad de codificar en BCD, sin más que considerar aisladamente cada dígito decimal según la Tabla 4.17. Los circuitos decodificadores y los algoritmos de transformación de código de E/S a BCD son muy sencillos.

Tabla 4.17. *Dígitos decimales codificados en binario.*

Dígito decimal	Valor binario
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

EJEMPLO 4.12

Representar en BCD el número decimal 98325

Consultando la Tabla 4.17:

9	8	3	2	5	=	BCD'1001 1000 0011 0010 0101
1001	1000	0011	0010	0101		

En la representación BCD de datos con signo se suelen utilizar cuatro bits para representar al signo. Puede ser, por ejemplo, 0000 para el signo positivo y 1001 para el signo negativo.

4.5.2 Datos de tipo real

Cuando se opera con números muy grandes se suele utilizar la notación exponencial. Según esta notación el número 13.257,3285, por ejemplo, puede representarse, entre otras, de las siguientes formas:

[4.15]

$$\begin{aligned} 13.257{,}3285 &= 13.257{,}3285 \cdot 10^{0} = 1{,}322573285 \cdot 10^{4} = \\ &= 0{,}132573285 \cdot 10^{5} = 132.573.285 \cdot 10^{-4} = \\ &= 13.257.328.900 \cdot 10^{-6} = \ldots \end{aligned}$$

Es decir que todo número, N, lo podemos expresar en la forma

[4.16]

$$N = M \cdot B^{E}$$

donde en el ejemplo (4.14) $B = 10$. Podemos transformar la representación del número N, conservando su valor, cambiando el exponente, E, y reajustando adecuadamente la mantisa, M: si aumentamos (disminuimos) en una unidad E, debemos dividir (multiplicar) M por B.

La **notación exponencial** también se conoce como **notación científica** o **notación en coma flotante**[2], ya que en (4.15) parece como si la coma decimal *flotase* de derecha a izquierda y al revés al cambiar el valor del exponente.

La representación y manejo de los datos puede ser responsabilidad del hardware del computador o de los traductores de lenguajes. Los microprocesadores actuales disponen internamente de un **procesador de coma flotante** (**FPU**, o *Float Point Unit)* que contiene circuitos aritméticos para operar con este tipo de datos. En la década de los noventa, cuando el microprocesador no incluía la FPU, existían coprocesadores aritméticos (por ejemplo, los circuitos integrados 8087, 80287 y 80387) que contenían la FPU. Si el hardware no dispone de circuitería para coma flotante, y un lenguaje de programación ofrece este tipo de datos, diversas rutinas de la biblioteca de programas del traductor correspondiente descomponen las operaciones en coma flotante en términos de las operaciones del lenguaje máquina, obteniéndose en este caso un rendimiento mucho menor en la ejecución de los programas.

Hasta la década de los años ochenta puede decirse que cada fabricante de computadores utilizaba un sistema propio para la representación de números reales; pero enseguida se observó la necesidad de algún sistema normalizado, ya que ello posibilitaría que se construyeran bibliotecas de programas de alta calidad, y que los diseñadores de computadores ideasen unidades aritmético-lógicas eficientes. Con este objetivo de 1977 a 1985 la asociación IEEE desarrollo un sistema normalizado de representación, denominado **Normalización IEEE 754** que afortunadamente hoy día tiene una aceptación prácticamente universal.

A continuación se presentará resumidamente la Normalización IEEE 754 (para más detalles pueden verse las referencias bibliográficas [COM81]). La forma de operar de la FPU con datos de coma flotante se estudiará en el contexto del Capítulo 9 (Sección 9.1.3).

4.5.2.1 Normalización IEEE 754 para la representación de datos de tipo real

Se trata de representar un número en la forma de la expresión [4.16], con E entero. La notación IEEE 754 establece las normas que se indican a continuación (Figura 4.10).

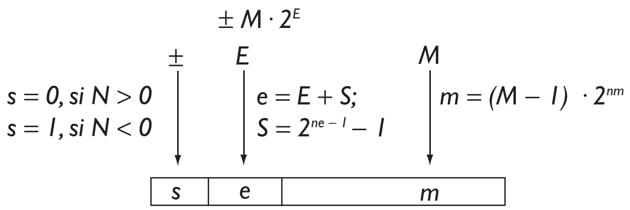


Figura 4.10. *Esquema de la representación IEEE 754.*

Elementos almacenados y orden de almacenamiento

La base del exponente es $B = 2$, es decir está predeterminada, por lo que sólo es necesario almacenar M y E, con sus signos respectivos. No se almacenan directamente el signo (+ o −), el exponente y la mantisa, sino que estos elementos sufren una transformación: realmente se memoriza lo que se denomina *campo del signo (s), campo del exponente* (o característica, e) y

[2] En los países sajones se denomina notación en **punto flotante** (*float point*), debido a que la parte entera la separan de la parte decimal con un punto en lugar de con una coma, como hacemos nosotros.

campo de la mantisa (m) que contiene las cifras significativas del número. Concretamente se utiliza un bit, *s,* como campo del signo del número, un número fijo de bits, *ne,* para almacenar el campo del exponente (incluyendo su signo), y otro cierto número fijo de bits, *nm,* para almacenar el campo de la mantisa. Es decir, siendo *n* el número total de bits utilizados para representar el número real, se verifica:

[4.17]

$$n = 1 + ne + nm$$

El orden de almacenamiento es campo de signo (*s*), campo de exponente (*e*) y campo de mantisa (*m*). es decir, el número quedaría representado como indica la Figura 4.10. Este orden se sigue para que los elementos y bits más significativos queden ordenados de izquierda a derecha, y así los algoritmos de comparación entre números enteros (ver si un número es mayor, igual, o menor que otro) sean también válidos para la representación de números reales.

Campo del signo

Cero para los números positivos y uno para los números negativos.

Campo del exponente

El exponente se almacena en la forma de *entero sesgado;* es decir, el exponente almacenado *e* se obtiene sumando al exponente del número, *E,* un sesgo *S*, dado por:

[4.18]

$$S = 2^{ne-1} - 1 \quad ; \quad e = S + E = 2^{n-1} + E - 1$$

de esta forma, en los *ne* bit reservados para el exponente se pueden incluir exponentes positivos o negativos sin utilizar un bit explicito de signo.

EJEMPLO 4.13

Supóngase que el exponente se almacena en 8 bits; es decir, ne = 8, y $S = 2^{ne-1} - 1$ = D'127 = B' 0111 1111, se producirán las siguientes transformaciones entre exponentes originales del número y exponentes almacenados:

Exponente (E)	*Exponente sesgado*	*Exponente almacenado (e)*
0	*127 + 0 = 127*	*0111 1111*
+ 2	*127 + 2 = 129*	*1000 0001*
+ 127	*127 + 127 = 254*	*1111 1110*
–1	*127 − 1 = 126*	*0111 1110*
–126	*127 − 126 = 1*	*0000 0001*

Campo de la mantisa

Por lo general el exponente se ajusta de forma tal que el 1 más significativo de la mantisa se encuentre en la posición 0 (posición de las unidades); es decir, $M \geq 1$. Cuando el número se encuentra ajustado de esta forma se dice que está **normalizado**, en caso de que no sea así se dice que está **denormalizado.**

Recuérdese que como la base del exponente es $B = 2$, si aumentamos una unidad el exponente, para que el valor del número no cambie, se debe dividir por 2 la mantisa binaria, o lo que es lo mismo, desplazar la coma decimal una posición a la izquierda, y viceversa. Por el contrario, si disminuimos una unidad el exponente, se debe multiplicar por 2 la mantisa, o lo que es lo mismo, desplazar la coma decimal una posición a la derecha, y viceversa, tal y como se pone de manifiesto en el ejemplo que se incluye a continuación.

EJEMPLO 4.14

Normalizar los números que se dan a continuación, suponiendo que las mantisas se dan en binario y los exponentes en decimal.

$$N1 = 1001{,}1100\ 110 \cdot 2^{-5} = 1{,}0011\ 1001\ 10 \cdot 2^{-2}$$

$$N2 = 0{,}0000\ 0110\ 1101 \cdot 2^{34} = 1.1011\ 01 \cdot 2^{28}$$

El campo de la mantisa se obtiene almacenando sólo la parte fraccionaria del número normalizado; es decir, no se almacena la información *'1.'*. Esto se hace así porque todos los números normalizados empiezan siempre por *1.*, por lo que se ahorra espacio de memoria no almacenando esta cabecera; diciéndose que el *1.* está *implícito,* u *oculto* o que el número se encuentra *empaquetado.* Obviamente cuando la ALU (o *FPU*) realice cualquier operación, debe restituir el *1.,* haciéndolo explicito; es decir, debe desempaquetar el dato. Analíticamente la condición de normalización puede expresarse así:

Reajustar mantisa y exponente de forma tal que se verifique:

$$1 \leq M < 2\ ;\ \ siendo\ M = [1, m] \qquad [4.19]$$

donde m es el valor que se almacena como mantisa.

EJEMPLO 4.15

Obtener los campos de mantisa de los datos indicados en el Ejemplo 4.14, suponiendo nm = 12 bits.

$N1 = 1{,}0011\ 1001\ 10 \cdot 2^{-2}$ luego: $m1 \leftarrow 0011\ 1001\ 1000$

$N2 = 1{,}1011\ 01 \cdot 2^{28}$ luego: $m2 \leftarrow 1011\ 0100\ 0000$

Las normalizaciones son necesarias para no perder precisión (cifras significativas) en operaciones sucesivas. Esto se debe a que el número de cifras binarias de la mantisa, *nm,* es fijo y conviene por tanto normalizar la mantisa de forma que las cifras más significativas ocupen las posiciones de mayor peso.

EJEMPLO 4.16

Suponiendo n = 16 bits, y ne = 8, obtener el valor decimal correspondiente al número, N, cuya representación interna es: NI = 1 0011 1110 0011 110

Los campos son:

$s = 1$, luego $N < 0$

$e = 0011\ 1110 = D'62$, luego $E = e - S = 62 - 127 = -65$

$m = 0011\ 110$, luego $M = B'1{,}0011\ 110 = 1 + 2^{-3} + 2^{-4} + 2^{-5} + 2^{-6} =$

$= 1 + 0{,}125 + 0{,}0625 + 0{,}03125 + 0{,}015625 = 1{,}234375;$

Con lo que: $N = -M \cdot 2^{E} = -1{,}234375 \cdot 2^{-65} = \mathbf{-3{,}345780142 \cdot 10^{-20}}$

Situaciones especiales (Figura 4.11)

a) Cuando el campo del exponente toma su valor mínimo; es decir, $e = 0$, el 1 más significativo de la mantisa no se encuentra implícito, y entonces la mantisa se almacena denormalizada. En este caso el sesgo es: $S = 2^{ne-1} - 2$; es decir, el valor del exponente correspondiente a los números denormalizados es: $E = e - S = -2^{ne-1} + 2$.

b) El número $N = 0$ se representa con todos los bits del campo del exponente y del campo de la mantisa cero; esto es: $e = 0$, $m = 0$.

c) Si todos los bits del campo del exponente son unos (es decir, éste adquiere su valor máximo), el dato representa:

- si $m = 0$, más o menos infinito (el resultado de dividir por 0, por ejemplo).
- si $m \neq 0$ representa un número indeterminado: el patrón de bits no representa a un número (es un código **NaN**, *No a Number*). Estos patrones de bits se utilizan para almacenar valores no validos (resultados de operaciones tales como $0 \cdot \infty$, $\infty \cdot \infty$, raíz cuadrada de un número negativo, etc.).

		Signo	Exponente	Mantisa
a) Número denormalizado	→	0	0000 0000	m ≠ 0
b) Cero	→	0	0000 0000	000 0000 0000 0000 0000 0000
c1) Más Infinito (+ ∞)	→	0	1111 11111	000 0000 0000 0000 0000 0000
c2) Menos Infinito (– ∞)	→	1	1111 11111	000 0000 0000 0000 0000 0000
c3) Número indeterminado (NaN)	→	0	1111 1111	m ≠ 0

Figura 4.11. *Patrones de bits asociados a situaciones especiales en la notación IEEE 754 (suponemos ne = 8 bits y nm = 23 bits).*

Redondeos

Un problema que se plantea al representar números reales es que, a diferencia de los números enteros, por lo general un numero decimal real, incluso aunque tenga un número finito de cifras significativas, no puede ser representado exactamente con un número fijo, *nm + 1,* de cifras binarias, lo que implica tener que utilizar técnicas de **redondeo**. La ALU o la FPU realiza las operaciones con más bits de los utilizados para representar los datos, de forma que al concluir la operación y tener que proporcionar un resultado final para almacenar debe también realizar un redondeo.

Los redondeos implican aproximar un número de una precisión determinada por otro de menor precisión. Obviamente en esta aproximación se comete un error. Hay dos alternativas básicas para redondear un número determinado: por defecto y por exceso. En la primera opción se trunca el número, no considerando las cifras en posiciones inferiores a la menos significativa a conservar, y en el segundo caso se suma una unidad a la cifra menos significativa a conservar. Para aproximar un número a otro usualmente se utiliza la técnica del **redondeo al más próximo**, según la cual se redondea por defecto o por exceso de forma tal que el valor absoluto del error de aproximación sea el menor. Aplicando este criterio a números decimales, si la primera cifra significativa que se desprecia (denominada *cifra de guarda*), es mayor que 5, se redondea por exceso, y si es menor de 5 se redondea por defecto (se trunca); así el error siempre será el menor, como se pone de manifiesto en el siguiente ejemplo.

EJEMPLO 4.17

Suponiendo que se están realizando operaciones en euros, y como resultado del cálculo de los intereses a cobrar a distintos clientes se obtienen los siguientes resultados:

45,636 €, 37,573 €, 7,435 €

Calcular los errores cometidos al redondear por defecto y por e×ceso estas cantidades.

Obviamente, como la moneda más pequeña es de céntimos de euro, habrá que redondear a dos cifras decimales significativas. Las posibilidades de redondeo son las que se indican en la tabla siguiente:

Número original	Redondeo por defecto		Redondeo por exceso	
	Nª redondeado	Error cometido	Nº redondeado	Error cometido
45,636 €	45,63 €	0,006 €	**45,64 €**	0,004 €
37,573 €	**37,57€**	0,003 €	37,56 €	0,007 €
7,435 €	7,43 €	0,005 €	7,44 €	0,005 €

Se comprueba que el redondeo al más pró×imo (por defecto o por e×ceso, dependiendo si la cifra de guarda es menor o mayor de 5, respectivamente), conduce al error de apro×imación menor de las dos opciones.

En el tercer caso del ejemplo anterior aprecia que cuando la cifra de guarda es 5, se comete el mismo error si se redondea por defecto o por exceso. La costumbre usual es redondear, en este caso, por exceso; aunque esto no es justo, pues las personas que pagan cantidades resultarán siempre perjudicadas con respecto a las que cobran. En consecuencia, podemos decir que existe

una situación problemática o anómala cuando el error coincide con la mitad del valor del peso de la última cifra significativa representable. En los circuitos aritméticos de un computador el problema es de gran importancia, ya que normalmente los cálculos se suelen hacer en grandes series iterativas y los errores se pueden ir acumulando, produciéndose una tendencia a que los números aumenten a causa del redondeo citado. Además, en binario la probabilidad de que se plantee la situación anómala es muy elevada (un 25%, aproximadamente). En consecuencia los resultados serán incorrectos si se hace un redondeo como el indicado. Una decisión equitativa, en la situación anómala, consiste en elegir aleatoriamente si se redondea por defecto o por exceso. Es lo que hace el estándar IEEE.

La normalización IEEE 754 recomienda efectuar un **redondeo al más próximo**; y si el error es igual en ambos sentidos se hace un **redondeo al par**, que consiste en *redondear por defecto o por exceso, pero siempre de forma que el bit menos significativo del número resultante sea 0.* En otras palabras en el caso citado, si la cifra menos significativa que se retiene es 0: se trunca el número, y si es 1, se le suma un 1. Como, en general, existe igual probabilidad de que la cifra menos significativa sea 0 o 1, el sistema de redondeo al par es equitativo, ya que en la situación problemática, por término medio el 50% de las veces, se hace el redondeo por defecto y el otro 50% por exceso.

Hay que hacer notar que para aproximar un número binario, y hacer una estimación del error, se necesitan dos cifras fraccionarias menos significativas que la que se retiene (posición $-nm$); es decir además del **bit de guarda** (posición $-(nm + 1)$) la ALU o FPU considera el siguiente bit menos significativo, que se suele denominar **bit de redondeo** (posición $-(nm + 2)$). El siguiente ejemplo pretende aclarar los conceptos expuestos.

EJEMPLO 4.18

Supóngase que las mantisas de los datos se almacenan en 5 bits (nm = 5), indicar los números resultantes de efectuar el redondeo según el estándar IEEE 754 de los números que se indican en la primera columna de la siguiente tabla.

Resultado en la ALU	*Acción*	*Mantista redondeada*	*Comentario*
1,01101 00	*Truncar*	*1,01101*	*Redondeo al más próximo*
1,01100 00	*Truncar*	*1,01100*	*Redondeo al más próximo*
1,01101 01	*Truncar*	*1,01101*	*Redondeo al más próximo*
1,01100 01	*Truncar*	*1,01100*	*Redondeo al más próximo*
1,01101 10	*Sumar 0,00001*	*1,01110*	*Redondeo al par: como el bit –5 es 1, se suma 1*
1,01100 10	*Truncar*	*1,01100*	*Redondeo al par: como el bit –5 es 0, se trunca*
1,01101 11	*Sumar 0,00001*	*1,01110*	*Redondeo al más próximo*
1,01100 11	*Sumar 0,00001*	*1,01101*	*Redondeo al más próximo*

Obsérvese que cuando las cifras de guarda y de redondeo son 00 o 01 se trunca, ya que así el error será menor que si se suma 1 a la cifra menos significativa que se retiene (bit de posición –5); en el caso de que dichas cifras sean 11, se suma 1 a la cifra menos significativa que se

retiene (bit de posición –5), ya que así el error será menor que si se trunca. En el caso de que las cifras indicadas sean 10, el error es igual si se trunca que si se suma 1: en este caso hay que efectuar un redondeo al par; para ello analizamos el valor del bit menos significativo que se retiene del número original (bit de posición –5), y si es 1, se suma 1, y sino se trunca. Siempre en este último caso la cifra menos significativa del número final es 0, cifra considerada par y de ahí la denominación *redondeo al par.*

Precisiones

El estándar IEEE 754 considera cuatro tamaños o precisiones posibles de datos (cuanto mayor sea *nm,* mayor es el número de cifras significativas y por tanto mayor es la precisión): simple precisión *(n = 32),* simple ampliada, doble y doble ampliada; aunque el estándar sólo especifica completamente las precisiones sencilla y doble (véase Tabla 4.18), según se indica a continuación:

Tabla 4.18. *Tipos de precisión considerados en el estándar IEEE 754.*

	Precisión			
	Simple	**Simple ampliada**	**Doble**	**Doble ampliada**
nm + 1 (bits de precisión)	24	32	53	64
E(máx)	127	1023	1023	16383
E(mín)	–126	–1022	–1022	–16382
S (sesgo del exponente)	127	(n.e.)	1023	(n.e.)

(n.e.: no especificado por el estándar)

a) Simple precisión:

En este caso $n = 32$ bits, $ne = 8$ bits, $nm = 23$ bits. El sesgo que hay que sumar a los exponentes es, véase expresión [4.18]: $S = 2^7 - 1 = 127$. El valor numérico, *N,* que representa el número almacenado, *[s e m],* resumiendo lo dicho en los párrafos anteriores es:

a) si $e = 255$ y $m \neq 0$, N no representa un número, *(NaN)*
b) si $e = 255$ y $m = 0$, $N = (-1) \cdot \infty$ *(valores especiales)*
c) si $0 < e < 255$, $N = (-1)^S \cdot [1.m] \cdot 2^{e-127}$ *(número normalizado)*
d) si $e = 0$ y $m \neq 0$, $N = (-1)^S \cdot [0.m] \cdot 2^{-126}$ *(número denormalizado)*
e) si $e=0$ y $m=0$, $N = (-1)^s \cdot 0$ *(valor cero)*

b) Doble precisión:

En este caso $n = 64$ bits, $ne = 11$ bits, y $nm = 52$ bits. El sesgo que hay que sumar a los exponentes es, véase expresión [4.18]: $S = 2^{10} - 1 = 1.023$. El valor numérico, *N,* que representa el código almacenado, N_{ri} = [s e m], resumiendo lo dicho en los párrafos anteriores es:

a) si $e = 2047$ y $m \neq 0$, N no representa un número, *(NaN)*
b) si $e = 2047$ y $m = 0$, $N = (-1) \cdot \infty$ *(valores especiales)*
c) si $0 < e < 2047$, $N = (-1)^S \cdot [1.m] \cdot 2^{e-1023}$ *(número normalizado)*
d) si $e=0$ y $m \neq 0$, $N = (-1)^S \cdot [0.m] \cdot 2^{-1022}$ *(número denormalizado)*
e) si $e=0$ y $m=0$, $N = (-1)^s \cdot 0$ *(valor cero)*

Valores límite

Con toda representación se obtienen unos valores máximos y mínimos representables. Teniendo en cuenta las situaciones especiales y el número de bits de cada campo, podemos obtener los números máximos y mínimos representables. En la Figura 4.12 se indican los patrones correspondientes a los números mayores y menores, en precisión sencilla. A continuación, como ejemplo, vamos a obtener estos valores extremos para este caso. Recordemos que: $B = 2$, $nm = 23$, $ne = 8$ y $S = 127$.

a)	***Infinito***	→	0	1111 11111	000 0000 0000 0000 0000 0000
b)	***Número mayor, N (máx)***	→	0	1111 1110	111 1111 1111 1111 1111 1111
c)	***Número menor normalizado, N (mín, nor)***	→	0	0000 0001	000 0000 0000 0000 0000 0000
d)	***Número menor denormalizado, N(mín,den)***	→	0	0000 0000	000 0000 0000 0000 0000 0001
e)	***Cero***	→	0	0000 0000	000 0000 0000 0000 0000 0000

Figura 4.12. *Patrones para los límites de números positivos (mayores y menores) representables en IEEE 754, precisión sencilla.*

Valor del número mayor N(máx)

El número de valor absoluto mayor distinto de infinito, corresponde al patrón de la Figura 4.12b; es decir:

- mantisa mayor: $m(máx) = 0{,}1111 \cdots 11 = 1 - 0{,}0000 \cdots 01 = 1 - 2^{-nm} = 1 - 2^{-23} = 0{,}99999988$
  $M(máx) = 1 + m(máx) = 1 + 0{,}99999988 = 1{,}99999988$
- exponente mayor: $E(máx) = e(máx) - S = 11111110 - 0111111 = 01111111 = 127$
- número mayor: $N\ (máx) = M(máx) \cdot 2^{E(máx)} = 1{,}99999988 \cdot 2^{127} = 3{,}402823466 \cdot 10^{38}$

Valor del número más pequeño próximo a cero N(máx)

Para obtener el número de valor absoluto más pequeño distinto de cero debemos considerar dos casos, dependiendo de si el número está normalizado o no.

a) *Número más pequeño normalizado*: corresponde al patrón de la Figura 4.12c; es decir:
   - mantisa menor normalizada:
     $m(mín) = 0{,}000 \cdots 00 = 0$
     $M(mín) = 1 + 0 = 1$
   - exponente menor normalizado: $E(mín) = 0000\ 0001 - 0111111 = 1 - 127 = -126$
   - número menor normalizado:
     $N\ (mín,nor) = M(mín) \cdot 2^{E(mín)} = 1 \cdot 2^{-126} = 1{,}175 \cdot 10^{-38}$

b) *Número más pequeño denormalizado*: corresponde al patrón de la Figura Figura 4.12d; es decir:
   - mantisa menor normalizada:
     $N(mín) = m(mín) = 0{,}000 \cdots 01 = 2^{-23}$

- exponente de los números denormalizado: *E(den) = –126*
- número menor denormalizado:
  $N\ (mín,den) = M(mín) \cdot 2^{E(den)} = 2^{-23} \cdot 2^{-126} = 1{,}1401 \cdot 10^{-45}$

Los valores máximo *N(máx)* y mínimos *N(mín)* anteriores se tendrán tanto para números positivos como negativos (Figura 4.13).

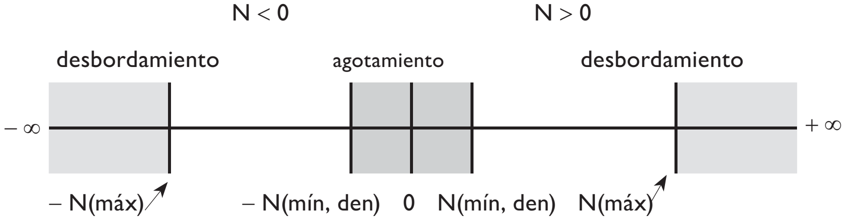


Figura 4.13. *Límites de números reales representables en IEEE 754.*

Obsérvese que los números reales que cumplan las siguientes condiciones no pueden ser representados:

- $-N(min,den) < N < N(min,den)$ *con* $N \neq 0$

  Si como resultado de una operación el número N tuviese que estar comprendido en esa zona, se dice que se produce un **agotamiento** (*underflow).* Las zonas de la recta real comprendidas entre:

  $N(min,deno) < N < N(min,nor)$ y $-N(min,nor) < N < -N(min,den)$

  corresponden a los datos denormalizados, denominándose estas zonas de **agotamiento gradual a cero.**
- $N < -N(máx)$ y $N > N(máx)$, con $N \neq \pm\infty$

  Si como resultado de una operación el número N tuviese que estar incluido en esa zona, se dice que se produce un **desbordamiento** *("overflow").*

Para tener una visión completa del proceso de transformación de un número real en la representación IEEE 754, a continuación se incluye un ejemplo.

EJEMPLO 4.19

Obtener la representación interna, N(ri) del número decimal $N = -543{,}7 \cdot 10^{-17}$*, según la norma IEEE 754 para datos en simple precisión*

Como se ha indicado en los párrafos anteriores, en la notación del IEEE 754: n = 32, ne = 8, nm = 23, B = 2, y la normalización se efectúa con el 1 más significativo en la posición de los enteros.

Para hacer la transformación el número original lo tenemos que poner en la forma (4.16) con B = 2:

[4.20]

$$N = -543{,}7 \cdot 10^{-17} = M \cdot 2^{E}$$

a continuación debemos normalizar la mantisa, pasar M y E a binario y sesgar el e×ponente. Para pasar M y E a binario es más cómodo utilizar la notación he×adecimal. También se simplifican los cálculos si antes de operar con el número original lo normalizamos.

EJEMPLO 4.19 (Cont.)

a) Transformación del número a la forma dada por la expresión [4.20],
Primero normalizamos el número original:

[4.21]

$$N = -543,7 \cdot 10^{-17} = -5,437 \cdot 10^{-15}$$

Para pasar a la forma (4.20), tenemos que tener en cuenta que:

[4.22]

$$10^{-15} = 2^{-x} \quad \Rightarrow \quad -15 \cdot \log(10) = x \cdot \log(2)$$

es decir:

[4.23]

$$x = -15 \cdot \frac{\log(10)}{\log(2)} = \frac{-15}{0,301029995} = -49,82892142$$

Como el exponente de 2, en la expresión [4.20], debe ser entero, la parte fraccionaría del valor de x *obtenido la debemos incluir en la mantisa inicial de N:*

[4.24]

$$N = -5,43,7 \cdot 2^{-49,82921427} = -(5,437 \cdot 2^{-0,8282142}) \cdot 2^{-49} = -3,060758897 \cdot 2^{-49}$$

b) Transformación de la mantisa a binario:
Utilizamos la notación he×adecimal como código intermedio. Como tenemos que obtener 26 bits (24 de la mantisa y 2 de redondeo) debemos calcular en total 26/4 = 6,5 cifras he×adecimales (en realidad 7):

3, =	3,	→ D'3 = H'6	→ 3,
0,060758897 × 16 =	0,972142348	→ D'0 = H'0	→ 3,0
0,972142348 × 16 =	15,55427758	→ D'15 = H'F	→ 3,0F
0,55427758 × 16 =	8,86844121	→ D'8 = H'8	→ 3,0F8
0,86844121 × 16 =	13,89505935	→ D'13 = H'D	→ 3,0F8D
0,89505935 × 16 =	14,32094966	→ D'14 = H'E	→ 3,0F8DE
0,32094966 × 16 =	5,135194522	→ D'5 = H'5	→ 3,0F8DE5

Es decir:

[4.25]

$$N = -3,060758897 \cdot 2^{-49} = -3,0F8DE5 \cdot 2^{-49} =$$
$$= -11,0000\,1111\,1000\,1101\,1110\,0101 \cdot 2^{-49} =$$
$$= -1,100\,0011\,1110\,0011\,0111\,100101 \cdot 2^{-48}$$

EJEMPLO 4.19 (Cont.)

donde hemos normalizado el número resultante. De la mantisa final, únicamente se utilizarán 23 bits de la parte fraccionaria, con lo que los dos últimos (01) corresponden a los bits de guarda y redondeo, respectivamente. Al ser estos bits 01, el redondeo al más próximo se realiza truncando. En consecuencia los 23 bits de la mantisa a almacenar son:

[4.26]

$$m = 100\ \ 0011\ \ 1110\ \ 0011\ \ 0111\ \ 1001$$

c) Exponente a almacenar:

Ahora debemos sesgar el exponente (−49) y pasarlo a binario:

[4.27]

$$e = E + S = -48 + 127 = D'79 = H'4F = B'0100\,1111$$

d) Número empaquetado

El número final, teniendo en cuenta que el número es negativo, será

signo	***exponente***	***mantisa***
1	010 01111	100 0011 1110 0011 0111 1001

o, abreviadamente:

A7C3E379

4.5.2.2 Consideraciones sobre la precisión

Un buen programador debe tener muy en cuenta cómo se almacenan los números reales en el computador, ya que a veces se pueden presentar problemas inherentes a representar los números con un número limitado de bits. Las dificultades pueden presentarse, entre otros, por los siguientes motivos:

a) Por la obtención, en resultados intermedios, de números excesivamente pequeños. Esto puede ocurrir por restar dos números muy iguales o por la división entre números en los que el divisor es mucho mayor que el dividendo. En estos casos puede perderse la precisión de los cálculos o, en el último, producirse un **agotamiento**.

b) Por la obtención de resultados numéricos excesivamente altos, es decir por **desbordamiento**. Esto ocurre, por ejemplo, al dividir un número por otro mucho menor que él o al efectuar sumas o productos sucesivos con números muy elevados.

c) En la comparación de dos números. Hay que tener en cuenta que en general las mantisas decimales no pueden representarse exactamente con un número (*nm)* limitado de bits, con lo que se genera un **error de representación**. Esto da lugar a problemas al comparar si un número es igual a otro, sobre todo si estos números se han obtenido por cálculos o procedimientos distintos. El computador considera que dos números son iguales únicamente si lo son *todos* sus bits. Para eludir este problema, el programador

(o compilador) debe detectar las igualdades utilizando números enteros o considerando que dos números son iguales si la diferencia entre ellos es menor que un valor dado.

d) Una consecuencia de los comentarios anteriores es que, como se puso de manifiesto en la Sección 2.1.2, la suma y multiplicación de datos de tipo real no siempre cumplen las propiedades asociativa y distributiva, pudiéndose obtener resultados distintos dependiendo del orden en que se realizan las operaciones.

4.6 Detección de errores

En la Sección 4.2 vimos que si codificamos cada símbolo de un conjunto dado por un número fijo de bits, para representar *m* símbolos distintos necesitamos al menos *n* bits, siendo *n* el menor número natural que verifica la relación (4.2), que repetimos por comodidad:

[4.28]

$$n \geq \log_2 m = 3{,}32 \cdot \log(m)\,,\ con\ n \in \aleph$$

También, en la sección mencionada, se ha indicado que a veces no se utilizan todas las combinaciones posibles de los *n* bits, ya que el número de símbolos a codificar *m* no tiene por qué ser igual a una potencia entera de 2. Cuantas menos combinaciones se desperdicien; es decir, no correspondan a símbolos a codificar, se dice que el código es más eficiente.

La **eficiencia de un código (τ)** se define como el cociente entre el número de símbolos que se representan realmente, *m,* dividido por el número, *m'*, de símbolos que en total pueden representarse; es decir, con códigos binarios en que $m' = 2^n$ se tiene:

[4.29]

$$\tau = \frac{m}{m'} = \frac{m}{2^n},\ \ con\ 0 \leq \tau \leq 1$$

Obviamente cuanto más eficiente *(mejor aprovechado)* sea el código, τ será mayor (se aproximará más a 1).

EJEMPLO 4.20

Supongamos que utilizamos el código ASCII básico (n = 7 bits), para representar 95 símbolos. ¿Qué eficiencia se obtiene con este código?, ¿y si se utilizase un bit más?

La eficiencia de esta codificación será:

$$\tau = \frac{m}{2^n} = \frac{95}{2^7} = 0{,}742$$

Si introdujésemos un bit adicional, la nueva eficiencia sería:

$$\tau = \frac{m}{2^n} = \frac{95}{2^8} = 0{,}371$$

Un código que es poco eficiente se dice que es redundante (más redundancias supone menos eficiencia), definiéndose la redundancia como:

[4.30]

$$R_d = (1 - \tau) \cdot 100\%$$

obsérvese que se da en tanto por ciento.

EJEMPLO 4.21

Obtener la redundancia para los casos considerados en el Ejemplo .

Aplicando la expresión [4.30]

[4.31]

$$R_d = (1 - 0,742) \cdot 100 = v25,8\%$$

$$R_d = (1 - 0,371) \cdot 100 = 62,9\%$$

Tabla 4.19. *Introducción de redundancias en un código.*

Alfabeto	Código I	Código II
A	000	1000
B	001	1001
C	010	1010
D	011	1011
E	100	1100
F	101	1101
G	110	1110
H	111	1111

A veces las redundancias se introducen deliberadamente para poder detectar posibles errores en la transmisión o grabación de información. Así, por ejemplo, si necesitamos transmitir 8 símbolos *{A, B, C, D, E, F, G y H}* y se hace con un código sin redundancias, necesitamos $n = 3$ bits, y un código posible puede ser el Código I de la Tabla 4.19. Si por algún error varía uno de los bits, obtenemos otro símbolo del alfabeto, que considerado por sí mismo (es decir, aisladamente) no puede ser detectado como erróneo. Si se utilizase un código redundante, tal como el Código II de la Tabla 4.19, existirían algunas posibilidades de detectar errores. Así, por ejemplo, si se transmite el símbolo H, esto es 1111, y por un error en la transmisión cambiase el primer bit, esto es, se recibiese 0111, podría detectarse el error ya que esta combinación no corresponde a ninguno de los símbolos posibles.

La introducción de redundancias con objeto de detectar o corregir errores se efectúa de acuerdo con algún algoritmo predeterminado; de esta manera los códigos pueden comprobarse

automáticamente por circuitos adecuados. Uno de los algoritmos más conocidos añade al código inicial de cada carácter un nuevo bit denominado **bit de paridad**. Existen dos criterios para introducir este bit:

- **Paridad par**: se añade un bit (0 o 1) de forma tal que el número total de unos de código que resulte sea par.
- **Paridad impar**: se añade un bit (0 o 1) de forma tal que el número total de unos del código que resulte sea impar.

  El ejemplo que se da a continuación aclara la cuestión.

EJEMPLO 4.22

Insertar un bit de paridad en los datos que aparecen en la primera columna de las siguientes tablas.

Mensaje o código inicial	***Mensaje o código con bit de paridad (criterio par)***
100 0001	**0**100 0001
010 1111	**1**010 1111
110 1000	**1**110 1000
111 0111	**0**111 0111

Bit de paridad

Mensaje o código inicial	***Mensaje o código con bit de paridad (criterio impar)***
100 0001	**1**100 0001
010 1111	**0**010 1111
110 1000	**0**110 1000
111 0111	**1**111 0111

Bit de paridad

El bit de paridad se introduce antes de transmitir (Figura 4.14) o grabar la información (en la memoria principal, cinta o disco). Por ruido o interferencias en la transmisión o defecto del soporte de información puede cambiar un bit (de 1 a 0 o de 0 a 1). Si en el receptor o al leer la información se comprueba la paridad del mensaje (esto es, se cuenta su número de unos), se detectaría el error, ya que el número de unos dejaría de ser par (en el caso de criterio par) o impar (en el caso de criterio impar). La introducción del bit de paridad y la comprobación se puede realizar con circuitos electrónicos muy sencillos (véase Capítulo 5, Ejemplo 5.1 y Ejercicios 5.17

y 5.18). En el caso de transmisión de datos, al detectar el receptor un error automáticamente solicita al emisor una nueva transmisión del mensaje erróneo, cosa que efectuaría éste también automáticamente. Obviamente, si se produjese el cambio simultáneo de dos o un número par de bits distintos no se detectaría el error de paridad; ahora bien, esta eventualidad es mucho menos probable que la de que cambie un sólo bit.

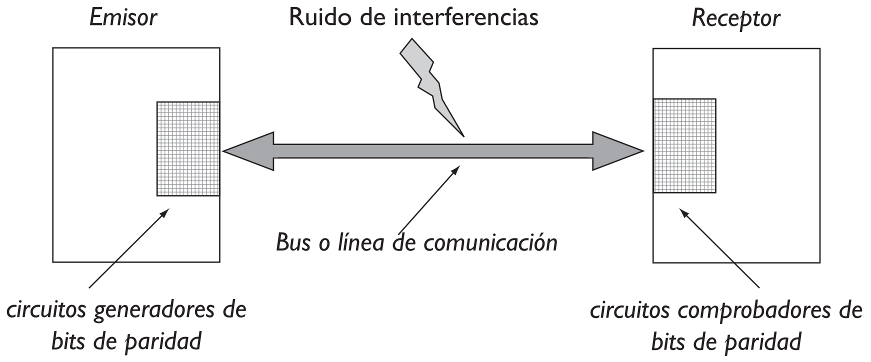


Figura 4.14. *Ejemplo de introducción de bit de paridad en una transmisión de información digital.*

Existen otros sistemas de introducir redundancias para poder detectar errores. Uno de ellos es el denominado de **verificación de cuenta fija**. En este tipo de sistema todos los códigos correspondientes a los caracteres tienen un número fijo y predeterminado de unos y ceros, siendo la posición relativa de los mismos la que determina el carácter a que corresponden. Por ejemplo, si quisiésemos codificar 70 símbolos distintos podríamos utilizar 8 bits de los cuales siempre 4 deben ser unos y los otros 4 ceros. Los circuitos de verificación tendrían que detectar en recepción o lectura si el número de unos (o ceros) de cada carácter coincide con el número prefijado.

EJEMPLO 4.23

Establecer un código de cuenta fija para codificar seis símbolos: A, B, C, D, E, F.
En la Tabla 4.20 se muestra una posible codificación utilizando n = 4 bits y siendo siempre el número de unos y ceros igual a 2.

Tabla 4.20. *Ejemplo de código de cuenta fija.*

Símbolo	*Código*
A	0011
B	0101
C	1001
D	0110
E	1010
F	1100

En el Capítulo 18 se analizan otros códigos (paridad vertical y códigos polinómicos) muy utilizados en transmisión de datos.

Hay que indicar que además de códigos detectores de errores, existen **códigos correctores de errores** con los que se introducen suficientes redundancias para, una vez detectado el error en un mensaje o carácter recuperar a partir de la información errónea recibida el mensaje o carácter original. Uno de los sistemas de codificación de este tipo más conocidos es el **código Hamming** [Leo01, Lin83]. Hay que hacer notar que existen técnicas de detección de errores para grabación de CD–ROM que permiten reducir la probabilidad de errores a un error en 20.000 discos.

También se definen códigos y transformaciones de códigos que permiten **cifrar** o **encriptar la información**, de forma tal que no pueda ser descifrada o comprendida por intrusos [Sch96, Sta99].

4.7 Compresión de datos

En muchas aplicaciones se requiere la utilización de archivos de gran capacidad. Así ocurre con los archivos de imágenes en aplicaciones multimedia (véase Ejemplo 4.8). En estos casos el tiempo de transmisión de una imagen por una red o el volumen requerido para su almacenamiento en disco pueden llegar a ser muy elevados. Este problema se presenta de forma notable en las aplicaciones multimedia, donde es frecuente utilizar combinadamente textos, sonidos, e imágenes tanto estáticas como dinámicas.

Existe un conjunto de técnicas para reducir el tamaño de un archivo. La transformación que se realiza se denomina **compresión de datos**. Según se indica en la Figura 4.15, el archivo, antes de ser almacenado o transmitido se comprime mediante un algoritmo de compresión, y cuando se recupera para procesarlo o visualizarlo se aplica la técnica inversa para descomprimirlo (**decompresión de datos**). Es conveniente señalar que habitualmente:

1) se utilizan simultáneamente varias técnicas superpuestas para poder obtener altos grados de comprensión
2) la compresión se reduce a recodificar la información representada internamente.

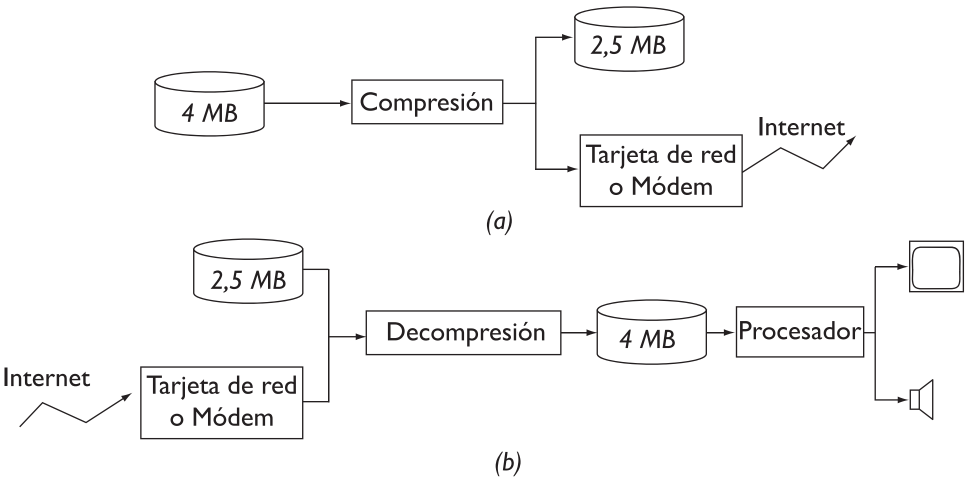


Figura 4.15. *Esquema de los procesos de compresión y de decompresión de un archivo.*

Hay algoritmos que se dice realizan una **compresión sin perdidas**, esto ocurre cuando la información original se comprime sin perder nada de la información inicial de forma tal que se puede recuperar exactamente la información de partida, sin perdida de calidad. A veces, para conseguir mayor compresión se toleran perdidas de calidad, no pudiéndose en la decompresión recuperar exactamente la información original, se dice entonces que se produjo una **compresión con perdidas**.

Denominando C_a y C_d a las capacidades del archivo antes y después de comprimirlo, respectivamente, podemos definir el **factor de compresión**,

[4.32]

$$f_c = \frac{C_a}{C_d}$$

Diciéndose que el archivo tiene una:

"*compresión de f_c a 1*" o *f_c:1.*

También a veces se da el **porcentaje de conversión** que indica el tanto por ciento que queda de la capacidad original después de la conversión; es decir:

[4.33]

$$p_C = \frac{C_d}{C_a} \cdot 100\%$$

Puede observarse que se verifica:

[4.34]

$$p_C = \frac{1}{f_c} \cdot 100\%$$

EJEMPLO 4.24

Suponiendo que un archivo de una capacidad de 4 MB se comprime ocupando 2,5 MB, obtener su factor y porcentaje de compresión.

SOLUCIÓN

Aplicando la expresión (4.32) se tiene:

$$f_c = \frac{C_a}{C_d} = \frac{4}{2,5} = 1,6$$

Es decir el archivo tiene una compresión de 1,6:1.

El porcentaje de compresión es:

$$f_C = \frac{C_d}{C_a} \cdot 100\% = \left(\frac{1}{f_c}\right) \cdot 100 = \left(\frac{1}{1,6}\right) \cdot 100 = 62,5\%$$

Es decir, el archivo original se ha reducido al 62,5% de su capacidad inicial.

A continuación se describen los fundamentos de algunas técnicas de compresión.

Codificación por longitud de secuencias (o RLE, *Run Length Encoded*)

Recuérdese que la información está representada internamente por ceros y unos. Una forma sencilla de comprimir los datos, en el caso de que en el archivo se repitan largas secuencias de ceros y unos, es sustituir cada una de éstas por el símbolo de la secuencia seguido por el número de veces que se repite en ella. Así, la secuencia de 9 bytes *RRRRRRRRR* ocupará solo 2 bytes si se almacena como *R9* (se produce una compresión 4,5:1). Este tipo de compresión resulta útil en archivos en los que se repiten largas secuencias; por ejemplo, el fondo de en una imagen de un espacio interior (la pared de una habitación) es muy continúo y queda, por tanto, representada por grandes secuencias de símbolos iguales.

Codificación relativa o incremental

Con frecuencia en imágenes y señales de sonido, los valores de dos píxeles consecutivos o de dos muestras de voz consecutivas en el tiempo difieren en una cantidad pequeña, en comparación con sus valores absolutos. Por ejemplo, las variaciones de color y de textura dentro de los objetos de una imagen suelen ser muy pequeñas, teniéndose sólo grandes variaciones en los contornos de dichos objetos. En este caso se logra una reducción considerable de almacenamiento si en lugar de almacenar los valores absolutos, se almacenan los incrementos respecto al valor anterior. Por ejemplo, si se tienen los siguientes valores consecutivos para una señal de audio:

155 156 160 174 180 179 170 175 177

con codificación relativa resultaría ser:

(155) 1 4 14 16 –1 –9 5 2

Obviamente los elementos de la segunda secuencia siempre van a ser números mucho más pequeños que los de la original y por tanto se pueden codificar con menos bits. Está técnica de compresión también se utiliza para imágenes en movimiento, cuando la variación entre imágenes sucesivas es muy pequeña. Este es el caso, por ejemplo, de imágenes de videoconferencias: una vez que se transmite completamente la imagen primera, sucesivamente sólo se transmiten las diferencias con respecto a la imagen anterior.

Codificación dependiente de la frecuencia

Consiste en representar cada símbolo con un código binario cuya longitud sea inversamente proporcional a la frecuencia; de esta manera los símbolos que se repiten más en un archivo se representarán con menos bits que los símbolos que aparecen con menos frecuencia. Los códigos de textos vistos en la Sección 4.2 se dice que son de longitud fija porque todos los símbolos se representan con un número fijo de bits: $n = 7$ en el caso de ASCII básico, $n = 8$ en Latín–1, $n = 16$ en Unicode, etc. En español las letras que más se repiten, además del espacio en blanco que separa a dos palabras consecutivas, son: *e, a, o, s;* podríamos asignar a estas letras códigos de 2 bits, por ejemplo: 00, 01, 10, y 11, respectivamente, y así sucesivamente para todos los caracteres. Los caracteres menos probables (*w, k,* x, *z, j*) tendrían los códigos más largos. De esta forma, al codificar un texto, se ahorraría gran cantidad de espacio; eso sí, a costa de aumentar las dificultades de procesamiento. La compresión consistiría en recodificar el texto con el código de longitud variable, y la descompresión recuperar el texto original en código de longitud fija.

Codificación con diccionario adaptativo

La idea básica consiste en realizar un diccionario con secuencias de bits, y sustituir en el archivo a comprimir dichas secuencias por su índice (número de orden) dentro del diccionario. El diccionario es adaptativo en el sentido de que según se va procesando la información, para comprimirla se va creando el diccionario: dada una secuencia determinada, primero se consulta si ya está en el diccionario; si es así, se sustituye por el índice correspondiente; si no, la nueva secuencia se incluye en el diccionario y se sustituye en el archivo por el nuevo índice. La compresión se obtiene debido a que siempre ocuparan menos los índices que las secuencias que representan. El archivo se forma conteniendo al principio (o final) del mismo el diccionario y luego (o antes) el contenido de la información del usuario representada por sus índices.

EJEMPLO 4.25

Decodificar el mensaje que se da a continuación, usando el siguiente código, dependiente de la frecuencia: A → 10 B → 11 C → 0

1011001011011001100 10010

Para llevar a cabo la decompresión vamos extrayendo de forma ordenada los siguientes grupos de bits: {10}, {11} y {0}, que corresponden a los símbolos A, B y C, respectivamente:

Mensaje comprimido →	10	11	0	0	10	11	0	11	0	0	11	0	0	10	0	10
Mensaje decomprimido →	A	B	C	C	A	B	C	B	C	C	B	C	C	A	C	A

Obsérvese que no es necesario ningún carácter separador entre los códigos de cada letra, no produciéndose ninguna ambigüedad. Este tipo de codificación se denomina codificación de Huffman.

Codificación Lempel-Ziv LZ77

Es un caso particular de diccionario adaptativo, donde se busca si los siguientes caracteres a comprimir han aparecido previamente en una secuencia anterior, caso de ser así esa cadena de caracteres se codifican con 3 números *(m, n, s),* donde *m* representa el lugar hacia atrás donde se inicia la secuencia previa encontrada, *n* es la longitud de la secuencia y *s* es el siguiente carácter de la cadena comprimida. Se observa que en este caso realmente no se crea un diccionario de cadenas. El siguiente ejemplo pretende aclarar el proceso.

En el ejemplo la información comprimida ocupa más espacio que la descomprimida; esto sencillamente se debe a que hemos puesto un ejemplo con tan sólo 35 bits y por tanto se generan secuencias muy cortas. Según aumenta el tamaño del archivo mayor probabilidad existe de encontrar secuencias previas más largas. En definitiva, la técnica consiste en encontrar la mayor secuencia previa que concuerde con los siguientes símbolos a comprimir del resto del archivo. Los conocidos programas **zip** y **unzip** para comprensión y decompresión de datos utilizan procedimientos de este tipo.

EJEMPLO 4.26

Supóngase que la siguiente sucesión de tripleta de números corresponden a un archivo comprimido por el sistema LZ77:

(0,0,0)(0,0,0)(1,1,1)(3,3,0)(8,3,0)(7,2,1)(2,2,1)(13,3,1)(10, 8,1)(5,3,0)

(Los paréntesis se incluyen para facilitar la comprensión del lector)

¿Cuál es la información original del archivo descomprimido?

Aplicando el algoritmo descrito, a información descomprimida será:

Compresión →	(0,0,0)	(0,0,0)	(1,1,1)	(3,3,0)	(8,3,0)	(7,2,1)	(2,2,1)	(13,3,1)	(10, 8,1)	(5,3,0)
Decompresión →	0	0	01	0010	0000	011	111	0101	011111011	1100

donde, por ejemplo, la tripleta (0,0,0) añade sencillamente un bit 0 y para obtener la información generada por la tripleta (3,3,0), nos hemos desplazado 3 posiciones a la izquierda y hemos añadido los 3 bits que aparecen a continuación, que son 001, y después añadimos el bit último de la tripleta; es decir un 0, con lo que la tripleta (3,3,0) genera la cadena [0010].

Compresión GIF (imágenes)

Con 3 bytes (uno por cada color básico) se pueden obtener 2^{24} = 16,777.216 mezclas de colores distintas. Esas mezclas proporcionan una gama de colores extraordinariamente rica con *calidad fotográfica.* Sin embargo, podemos obtener una imagen de gran calidad sin necesidad de utilizar un número excesivo de dichas mezclas; así, usualmente con 256 mezclas distintas elegidas convenientemente de entre las 2^{24} posibles se obtiene una calidad más que aceptable. El conjunto de mezclas seleccionadas se denomina **paleta de color**, y cada una de ellas se puede codificar con 1 byte. El fundamento del sistema de compresión GIF (*Graphic Interchange Format,* Tabla 4.10) consiste en almacenar como atributo de cada elemento de imagen una de las combinaciones de la paleta (1 byte) en lugar de la mezcla concreta representada por 24 bits (3 bytes). En el caso del ejemplo considerado en la Sección 4.4 (Ejemplo 4.8) el archivo de una imagen XGA que ocupa 2,25 MB pasaría a ocupar 768 KB. En el momento de visualizar la imagen, con ayuda de una tabla que contiene los valores RGB de cada color de la paleta, se transforma el índice de color (1 byte) por la combinación concreta RGB de los tres colores básicos a que corresponde (3 bytes). Sin más que cambiar las 256 combinaciones de colores de la paleta se pueden cambiar los colores de la imagen. En definitiva, con este sistema se obtiene una comprensión de 3 a 1 quedando reducido el archivo a un 33,33% de su capacidad original.

Además de esta considerable reducción, se puede incrementar ésta aún más asignando a uno de los valores de la paleta el atributo *transparente*, para indicar que los elementos de imagen con este código deben visualizar el color de fondo. La codificación GIF es muy útil para representar imágenes sintéticas en movimiento, tal como las que se producen en videojuegos o dibujos animados.

Compresión JPEG (imágenes)

En la TV en color, en lugar de transmitir las intensidades de los tres colores básicos (x_R, x_G, x_B) de cada elemento de imagen se utilizan tres valores transformados, denominados **luminancia** (o brillo, x_Y) y **crominancias** (x_I, x_Q), relacionadas con (x_R, x_G y x_B) como sigue:

[4.35]

$$x_Y = 0,30 \cdot x_R + 0,59 \cdot x_G + 0,11 \cdot x_B$$
$$x_I = 0,60 \cdot x_R - 0,28 \cdot x_G - 0,32 \cdot x_B$$
$$x_Q = 0,21 \cdot x_R - 0,52 \cdot x_G + 0,31 \cdot x_B$$

En los receptores de color, a partir de los tres valores anteriores se recuperan los valores de las intensidades x_R, x_G, x_B de los colores básicos que se visualizan directamente (para más detalles puede verse la Sección11.3.2). En cambio en una TV en blanco y negro se utilizaba tan sólo la luminancia, x_Y , que visualiza los niveles de grises con una gran calidad.

Se ha observado que el ojo humano es más sensible a los cambios espaciales del brillo (luminancia) que del color (crominancias). Uno de los diversos sistemas de compresión desarrollado por el **JPEG** (*Joint Photographic Experts Group,* Tabla 4.10) se fundamenta en este hecho, de forma que se almacena el brillo de cada punto de imagen, pero sólo la media de las dos crominancias obtenida sobre, por ejemplo, cinco puntos de imagen adyacentes. De esta forma, en vez de almacenar por cada 5 puntos 5 · 3 = 15 valores se almacena 5 + 2 = 7 valores, obteniéndose así una compresión de 2,14 a 1, reduciéndose a un 47% en la capacidad necesaria para almacenar la imagen. Se obtendría un archivo más comprimido, pero de peor calidad, si se almacenas los valores medios de las crominancias de nueve puntos en lugar de las de cinco, de forma que podemos reducir más el tamaño pero siempre a costa de la calidad.

En la compresión JPEG, el sistema descrito anteriormente se suele utilizar combinadamente con los otros métodos básicos de comprensión descritos al inicio de esta sección. Así, con la codificación incremental se suelen obtener muy buenos resultados, ya que por lo general en una gran parte de una imagen los colores de puntos adyacentes varían muy poco, por lo que resulta más económico representar los colores de un elemento de imagen por sus incrementos respecto a los de uno adyacente que por sus valores absolutos. En total, puede lograrse representar cada elemento de imagen con 4 bits. Adicionalmente se consigue mayor comprensión superponiendo una codificación de longitud variable.

Compresión MPEG (imágenes)

El MPEG *(Motion Picture Experts Group)* ha definido un estándar ISO para imágenes en movimiento. Se fundamenta en el sistema JPEG, pero además una imagen dentro de una secuencia se representa con tan sólo los *cambios* con respecto a la imagen anterior. De esta manera se obtienen reducciones muy considerables en los tamaños de los ficheros correspondientes. Conviene indicar que la visualización de los archivos así comprimidos requieren de un programa (denominado genéricamente **visor**) que regenere las imágenes originales, y de un procesador lo suficientemente potente para que la descompresión se efectúe a una frecuencia mayor de 30 imágenes/segundo, con objeto de ser observadas con una sensación de continuidad.

Compresión MP3 (sonido)

Según se puso de manifiesto en el Ejemplo 4.5 (Sección 4.3), también los archivos de sonido ocupan gran capacidad: en principio la grabación de un minuto de música con calidad CD ocupa unos 10 MBytes. El problema no sólo se presenta a la hora de almacenar la información, sino también a la hora de transmitirla, por ejemplo a través de Internet. Una de las técnicas más usadas para compresión de sonido se denomina MP3 (MPEG-1 Nivel 3 de audio) fue propuesta por grupo MPEG y es un estándar ISO. Las ideas básicas son las siguientes:

- Descomponer, mediante filtros, la señal de audio original en diversos canales de frecuencia (32, por ejemplo),
- Codificar la información de cada canal con un número de bits dependiente de la amplitud de la señal. Si la amplitud máxima de un canal es menor que la de otros, se codifica con un número menor de bits manteniendo la resolución auditiva.
- También el oído humano es más selectivo en unas bandas de frecuencias que en otras. Los canales para los que es menos selectivo se pueden codificar con menos resolución (menos bits)

- Se forman tramas de información con una cabecera de sincronización, bits detectores de error (CRC), número de bits de cuantización, factor de escala, las muestras y datos auxiliares.
- Se multiplexa la información de los distintos canales, y se almacena o transmite
- Con este sistema se obtienen porcentajes de comprensión del 12:1; es decir el archivo original queda reducido al 8,33 de su capacidad inicial.

EJEMPLO 4.27

A través de Internet se esta recibiendo una emisora de radio que se escucha con una calidad CD. Sabiendo que la tasa de datos con que se está recibiendo la emisión es de R = 16 Kbits/s, determinar el factor de compresión que se está utilizando.

Al tener una calidad CD, se verifica que la frecuencia de muestreo es: F_s = 44,1 KHz y entre muestra y muestra, en principio (sin comprimir), habría que transmitir

$$n = 16 \cdot \frac{bits}{muestra} \cdot 2\ canales = 32\ bits$$

es decir, sin compresión, la tasa de bits hubiese sido:

$$R_{sc} = (44{,}1 \cdot 32)\frac{Kb}{s} = 1.411{,}2\ \frac{Kb}{s}$$

pero como en realidad es: 16 Kb/s; el factor de compresión será:

$$f_C = \frac{1.411{,}2}{16} = 88{,}2$$

Es decir, hay una compresión de 88:1, o

$$p_C = \frac{1}{f_c} \cdot 100 = 1{,}13\%$$

Es decir, el número de bits transmitidos por segundo con la compresión es tal sólo un 1,13% de los que hubiese que transmitir sin ella.

Las señales de voz o música se graban o transmiten en formato MP3, ocupando un espacio reducido, y el sistema de reproducción de sonido contiene un procesador (circuito integrado) o CODEC lo suficientemente rápido para reconstruir la señal original en tiempo real para poder escucharla adecuadamente.

4.8 Conclusión

En este capítulo se han presentado los aspectos fundamentales relacionados con la representación de la información en el interior de los computadores.

Usualmente la información que se intercambia con el computador se encuentra bajo la forma de textos, o sonidos, o imágenes, por lo que en este capítulo hemos considerado la representación de la información independientemente en esos tres casos.

También hemos analizado el caso de los datos numéricos (números enteros o reales), ya que requieren una representación especial para poder operar matemáticamente con ellos de forma eficiente.

Directamente relacionados con la representación de la información se encuentran los conceptos de detección automática de errores en almacenamiento o transmisión, y de compresión de datos. Las dos últimas secciones de este capítulo han hecho una introducción a ellos.

Consideramos que con el estudio de este capítulo el lector habrá adquirido una idea clara sobre la representación de la información en el interior del computador, y, en general, en cualquier sistema digital de procesamiento o transmisión de información.

EJERCICIOS

4.1 ¿Cuántos bits se necesitarían como mínimo para codificar un conjunto de 108 caracteres?

4.2 Transformar a códigos ASCII, Latín-1, EBCDIC y Unicode los siguientes mensajes:
a) 37,45.
b) 7784 BCR.

4.3 ¿Qué velocidad de transmisión se necesitará para transmitir, sin compresión, una señal de audio de calidad radio, no estéreo?

4.4 ¿Qué tiempo de música en calidad CD estéreo se puede almacenar, sin compresión, en un CD-ROM de 650 MBytes?, ¿y en un DVD de 4,7 GBytes?

4.5 Cuántas imágenes caben en una memoria flash de 256 MBytes, suponiendo:
a) Resolución VGA.
b) Resolución SVGA.
c) Resolución XVGA.

4.6 Cuántas imágenes caben en un CD de 600 MBytes, suponiendo:
a) Resolución VGA.
b) Resolución SVGA.
c) Resolución XVGA.

4.7 Cuántas imágenes caben en un DVD de 4,7 GBytes, suponiendo:
a) Resolución VGA.
b) Resolución SVGA.
c) Resolución XVGA.

4.8 Qué tiempo se puede almacenar en en una memoria flash de 256 MBytes:
a) de imágenes de TV.
b) de imágenes de HDTV.

4.9 Qué tiempo se puede almacenar en un CD de 600 MBytes:
a) de imágenes de TV.
b) de imágenes de HDTV.

4.10 Qué tiempo se puede almacenar en un DVD de 17 GBytes:
a) de imágenes de TV.
b) de imágenes de HDTV.

4.11 Suponiendo un computador con longitud de palabra $n = 8$ bits y que utiliza representación en signo y magnitud, cómo se representarían internamente los siguientes números enteros: *65; –37; +84; –21*

4.12 Suponiendo un computador con longitud de palabra $n = 8$ bits y que utiliza representación en complemento a 1, ¿cómo se representarían internamente los siguientes números enteros: *65; –37; +84; –21?*

4.13 Suponiendo un computador con longitud de palabra $n = 8$ bits y que utiliza representación en complemento a 2, ¿cómo se representarían internamente los siguientes números enteros: *65; –37; +84; –2* ?

4.14 Suponiendo un computador con longitud de palabra $n = 8$ bits y que utiliza representación sesgada, ¿cómo se representarían internamente los siguientes números enteros: *65; –37; +84; –21*?

4.15 ¿Cuáles serían los números decimales enteros correspondientes a los números:
1010 1110; 0111 1011; 1000 0000
suponiendo las siguientes representaciones?:
a) sin signo,
b) signo y magnitud,
c) complemento a 1,
e) complemento a 2,
f) sesgada, y
g) BCD.

4.16 Suponiendo un computador de longitud de palabra de 16 bits y que utiliza el criterio del extremo menor, ¿cómo se almacenaría en memoria el dato $5_1454.327$ al utilizar las siguientes representaciones?:
a) sin signo,
b) signo y magnitud,
c) complemento a 1,
h) complemento a 2,
i) sesgada, y
j) BCD.

4.17 Repetir el ejercicio anterior suponiendo que el criterio de almacenamiento en memoria es del extremo mayor.

4.18 Suponiendo un computador que utiliza el estándar IEEE 754, obtener las representaciones internas en simple precisión de los siguientes números:

425372,27; –37.48; $-25{,}38 \cdot 10^{-17}$; 7; $1{,}5 \cdot 10^{-40}$

4.19 En un programa se define el número 3 como número entero (J = 3), y como número real en simple precisión (x = 3, 0). Suponiendo que en los dos casos los datos se almacenan con 32 bits, y que se los enteros se representan en complemento y los reales en la notación IEEE754, obtener las representaciones internas para las valores de J y x. ¿Cuál de las dos representaciones es más precisa?

4.20 En IEEE 754, precisión sencilla, se tienen los datos que se indican a continuación; ¿a que números decimales corresponden?:
a) BC7E 0000;
b) 3754 3000;
c) 00A0 0000;
d) 8000 0004;
e) FF80 0000;
f) 7FF8 0700

4.21 Obtener los números reales máximos y mínimos representables en IEEE 754, doble precisión.

4.22 Un computador de longitud de palabra de memoria de 16 bits, criterio extremo mayor, utiliza el código ASCII con paridad impar para entrada/salida de los datos. Suponiendo que con el editor de C se introduce el dato x = 3,2:
a) ¿Cómo queda inicialmente almacenado ese dato en la memoria?
b) Suponiendo que x se define como dato en simple precisión y se utiliza la notación IEEE 754, ¿cómo lo almacenará en memoria el compilador de C?

4.23 ¿Cuál será la eficiencia mínima obtenible al codificar 108 símbolos en binario? ¿qué redundancia se obtendría? ¿Cuál sería la eficiencia y redundancia si se utilizasen dos bits más de los necesarios?

4.24 ¿Qué eficiencia y redundancia tiene la codificación EBCDIC dada en la Tabla 4.3?

4.25 ¿Qué eficiencia y redundancia tiene la codificación ISO 8859-1 (Latín 1) dada en la Tabla 4.6?, ¿y si añadiese a cada carácter un bit de paridad?

4.26 Obtener el bit de paridad, según el criterio impar, de los siguientes mensajes:
10010 ; 10000 ; 101111 ; 1010

4.27 En el interior de un chip de memoria se leen los datos que se dan a continuación. que incluyen un bit de paridad (criterio par):
1 1001 0010 0011 1001; 0 1110 0011 0011 1111; 0 1111 0011 1100; 0 1011 0000 1111 0101
Indicar cuáles de los datos anteriores se han grabado o leído erróneamente.

4.28 Un computador recibe de un terminal los siguientes caracteres ASCII, que contienen un bit de paridad (criterio impar):
7A; 5C; 47; CA; 7C; C8
Indicar cuáles de estos caracteres deben ser rechazados por ser erróneos.

4.29 ¿Qué patrón de bits se generarían al teclear el dato 45,32 si se utiliza el código ASCII (7 bits) con bit de paridad, criterio impar?

4.30 Se tiene un archivo grabado con codificación relativa. Si el valor inicial es 17.324, y la secuencia de valores que le siguen es:
5, –3, 18, 100, –32, 59, 27, 32, –5, 49
Obtener los valores originales (absolutos) del archivo.
Si los valores de la información original están en el rango ±32.767 y los valores incrementales en el rango ±127; obtener el factor de compresión logrado con la codificación relativa.

4.31 Como quedaría comprimida en Lemped-Ziv LZ77 la siguiente secuencia:
00010010000001111101010111110111100

4.32 Dentro de mi PC tengo grabado, con calidad CD, el Allegro del Concierto de Piano y Orquesta número 1 de Mozart. Se encuentra comprimido en MPEG 1 nivel 3, con un factor de compresión 7,35:1 (13,6%) y ocupa 7,44 MB. Obtener:
a) La capacidad del archivo descomprimido.
b) La tasa de datos en Kbps.
c) la duración que tendrá la audición del citado archivo.

4.33 ¿Qué factor de compresión habría que utilizar para grabar *Rigoletto* de Giuseppe Verdi en un disquete de 1,44 MBytes?; ¿y en un CD de 600 MBytes?, ¿y en dos CD? *(Ayuda: la versión a grabar de la opera indicada tiene una duración de 1 hora 55 minutos)*.

BIBLIOGRAFÍA

[Bro03] Brookshear, J.R.: *Computer Science*; 7th Edt. Addison Wesley, 2003.

[COM81] Sobre la notación IEEE 754 pueden verse, en la revista "Computer IEEE", Vol. 14, Nº 3, March 1981, los siguientes artículos:
- Cody, W.J.: *Analysis of proposals for the floating-point standard.*
- Coonen, J.T.: *Underflow and the denormalized numbers*
- Hough, D.: *Applications of the proposed IEEE 754 standard for floating-point arithmetic.*
- Stevenson, D.: *A proposed standard for binary floating-point arithmetic.*

[Leo01] Leon-Garcia, A.; Widjaja,I.: *Redes de comunicación. Conceptos fundamentales y arquitecturas básicas;* MacGraw-Hill, 2001.

[Fer98] Fernández, G.: *Conceptos básicos de arquitectura y sistemas operativos;* Sistemas y Servicios de Comunicación, 1998.

[Ham03] Hamacher, C; Vranesic,Z.; Zaky,S.: *Organización de computadores*, McGraw-Hill, 2003.

[Lin83] Lin, S.; Costello,D.J.: *Error Control Coding: Fundamentals and Applications;* Prentice Hall, 1983.

[Ort05] Ortega, J.; Anguita,M.; Prieto,A.: *Arquitectura de Computadores*, Thomson, 2005.

[Pri05] Prieto, A.; Prieto,B.: *Conceptos de Informática,* Serie Shaum, McGraw-Hill, 2005.

[Sch96] Schneider, B.: *Applied Cryptography,* Wiley, 1996.

[Sta06] Stallings, W.: *Organización y Arquitectura de Computadores*, 7ª Edición, Pearson Prentice Hall, 2006

[Sta99] Stallings, W.: *Cryptography and Network Security,* Prentice Hall, 1999.

[Tan00] Tanenbaum, A.S.: *Organización de computadoras. Un enfoque estructurado,* 4ª Edc. Pearson Educación, 2000.

CAPÍTULO 5

Sistemas digitales combinacionales

Este capítulo es una introducción al diseño de sistemas digitales. En primer lugar se presenta el álgebra de conmutación y los mapas de Karnaugh como herramientas útiles en el proceso de diseño. A continuación se analizan diferentes bloques combinacionales, como son los codificadores, decodificadores, multiplexores, etc., disponibles en forma de circuitos integrados. Posteriormente se describen diferentes bloques programables (memorias ROM, PALs y PLAs) disponibles para la síntesis de circuitos digitales. Con estos conocimientos se puede abordar el diseño de pequeños sistemas combinacionales.

El próximo capítulo se dedica al diseño de sistemas secuenciales. Ambos Capítulos 5 y 6 deben considerarse como una unidad. Con ellos se cubre el nivel de diseño lógico y con ellos se pretende dar una visión global del objetivo de este nivel descriptivo y de los elementos constructivos con los que se materializa cualquier sistema digital, entre los que se encuentran los computadores.

5.1 Sistemas digitales

Las magnitudes físicas usuales, tales como la presión y la temperatura, pueden tomar cualquier valor dentro de un rango de valores y sus variaciones pueden ser muy pequeñas (tanto como se quiera o se pueda apreciar). Por ejemplo, la temperatura en una habitación podría estar, pongamos por caso, entre –10ºC y +40ºC, tomando cualquier valor entre estos extremos, y en el intervalo de un minuto podría variar un grado, o una décima de grado, o una milésima de grado. Estas magnitudes físicas, que pueden tomar infinitos valores y varían de forma continua, se conocen como variables **analógicas** o **continuas**. Las señales físicas que en cada caso se utilicen para representar las variables analógicas se denominan **señales analógicas**. Por ejemplo, un sensor de presión puede generar una tensión eléctrica proporcional a la presión, de manera que la presión es una variable analógica y la tensión eléctrica generada por el sensor será una señal analógica. Los sistemas en los que se procesan señales analógicas se conocen como **sistemas analógicos**. Frente a ellos, como otra alternativa de representación, están los sistemas digitales.

Un **sistema digital** es cualquier dispositivo mecánico, eléctrico, electrónico, etc., destinado a la generación y/o representación y/o transmisión y/o procesamiento y/o almacenamiento de la información en el que dicha información esté representada por magnitudes físicas

(conocidas como señales) limitadas a tomar sólo unos determinados valores discretos. Por ejemplo, el conmutador con el que encendemos o apagamos la luz de nuestra habitación se puede considerar como un sistema digital. En este caso la magnitud física que se utiliza para representar la información es la resistencia eléctrica entre los puntos A y B de la Figura 5.1a y b, que sólo puede tomar dos valores: cero (o muy pequeña) e infinito (o muy grande); estos dos valores corresponden al contacto cerrado y al contacto abierto, respectivamente.

Un sistema digital en el que sólo se utilicen dos valores discretos se conoce como **sistema binario**; usualmente estos dos valores discretos se representan como cero y uno (0 y 1). El conmutador eléctrico anterior es un sistema digital binario, y las dos situaciones posibles, contacto abierto y contacto cerrado, se pueden representar por 0 y 1, respectivamente (o al contrario, si así lo deseamos; no es más que un convenio). Supongamos que construimos el circuito eléctrico de la Figura 5.1c y d con un conmutador, una bombilla y una pila eléctrica. Cuando el conmutador está cerrado, por el circuito circula la intensidad i = I, la diferencia de potencial entre los extremos del conmutador es $v_2 = 0$, la diferencia de potencial entre los extremos de la bombilla es v_1 = V, y la bombilla luce (luminancia l = L); (véase Figura 5.1c). Cuando el conmutador está abierto, por el circuito no circula intensidad ($i = 0$), la diferencia de potencial entre los extremos del conmutador es v_2 = V, la diferencia de potencial entre los extremos de la bombilla es $v_1 = 0$, y la bombilla no luce (luminancia $l = 0$); (véase Figura 5.1d). Así pues, en este circuito se tienen las cuatro magnitudes físicas siguientes, que se comportan como magnitudes digitales binarias:

a) i, que puede tomar los valores 0 e I,
b) las diferencias de potencial v_1 y v_2, que pueden tomar los valores 0 y V, y
c) la luminancia l de la bombilla, que puede tomar los valores 0 y L.

Los dos valores que pueden tomar i, v_1, v_2 y l se pueden representar por 0 y 1.

Otro ejemplo paradigmático de sistema digital es cualquier computador, en el que todas las señales físicas que se utilizan para la representación o el procesamiento de la información son señales digitales binarias.

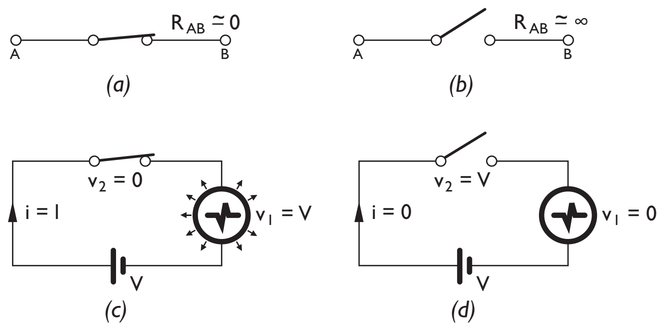


Figura 5.1. *El conmutador eléctrico como ejemplo de sistema digital: (a) contacto cerrado; (b) contacto abierto; (c) circuito eléctrico con un conmutador cerrado; (d) circuito eléctrico con un conmutador abierto.*

Los sistemas binarios son los sistemas digitales que se utilizan casi exclusivamente y son los únicos que en este texto se van a considerar, aunque los sistemas multivaluados (es decir,

con más de dos valores discretos) están siendo ampliamente investigados y se utilizan en la práctica como subsistemas de algunos sistemas binarios.

Cuando se dice que las señales están limitadas a sólo dos valores discretos, lo que realmente se quiere decir es que las señales pueden tomar valores dentro de dos rangos perfectamente diferenciables (que no se solapen, por ejemplo). Es decir, es perfectamente asumible una cierta variación de estas señales de sus valores nominales, siempre que estas desviaciones no lleven a confundir uno de los dos valores con el otro. Por ejemplo, para un chip podría ser aplicable el siguiente convenio: cualquier valor de tensión entre 0 voltios y 0,8 voltios en sus entradas o salidas corresponde al valor lógico 0; cualquier valor de tensión entre 2,4 voltios y 5,0 voltios corresponde al valor lógico 1; las tensiones entre 0,8 y 2,4 no corresponderían claramente a ninguno de los dos valores lógicos y sería un rango de valores a evitar.

Cualquier sistema digital, haciendo abstracción de su constitución, se puede representar esquemáticamente como un bloque con unas entradas $x_1 \dots x_n$, y unas salidas $z_1 \dots z_m$, tal como se hace en la Figura 5.2a, de manera que la acción del sistema digital consiste en asociar o asignar o producir una determinada combinación de salidas para cada combinación de entradas. Como ejemplo se puede considerar un sistema digital binario con tres entradas, x_1, x_2, y x_3, y dos salidas, z_1 y z_2 (véase Figura 5.2b); el funcionamiento de este sistema puede especificarse de una forma completa mediante la tabla de la Figura 5.2c. Dado que las entradas sólo pueden tomar los valores 0 y 1, para tres entradas hay ocho combinaciones diferentes de entradas, que aparecen numeradas de 0 a 7 en la Figura 5.2c. A cada una de las combinaciones de entradas le corresponde una combinación de salidas. Por ejemplo, a la combinación de entradas $x_1x_2x_3$ = 001 (fila 1) le corresponde la combinación de salidas z_1z_2 = 01. Este tipo de tablas, que describen de forma exhaustiva el comportamiento del sistema digital, se denominan tablas verdad. La tabla verdad correspondiente a un sistema digital con n entradas y m salidas tiene 2^n filas y $n + m$ columnas. Cada una de las filas corresponde a una combinación diferente de las entradas desde 00...0 hasta 11...1; las columnas de la 1 a la n corresponden a las n diferentes entradas; en cada una de estas n columnas hay 2^{n-1} ceros y 2^{n-1} unos, agrupados en 2, 4, 8... bloques, respectivamente, de manera que, por filas, se obtienen todos los 2^n valores posibles de las entradas. En las m últimas columnas se dan los valores que el sistema digital produce como salidas para cada uno de los diferentes valores de las entradas. En ocasiones se suele incluir al principio una columna extra en la que se da el número de orden de las filas, desde 0 hasta 2^n–1(así se ha hecho en la Figura 5.2c).

EJEMPLO 5.1

En un computador se va a procesar información en la que se manejan 16 caracteres diferentes y, por tanto, se necesitan como mínimo cuatro bits para codificar esa información. Supóngase que, para poder detectar errores, a esos cuatro bits se les va a añadir un bit de paridad par. Se trata de describir en forma de tabla verdad el comportamiento de un sistema digital para obtener ese bit de paridad par.

Solución: *el sistema digital tendrá cuatro entradas (los cuatro bits que se utilizan para codificar la información), sean x, y, z, u, generando como salida el bit de paridad par, llamémosle p. En la tabla de la Figura 5.2d se da el valor de p para todas y cada una de las combinaciones de valores de las entradas.*

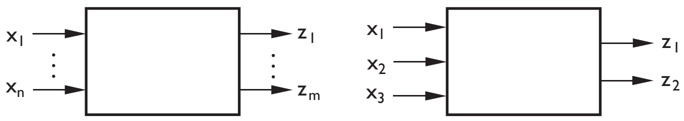
EJEMPLO 5.2

Describir en forma de tabla verdad el funcionamiento de un circuito incrementador/decrementador módulo 8 que tenga como entrada un número binario de tres bits, E = xyz, más una entrada de control, c, y cuya salida sea también un número binario de tres bits, S = uvw. La función que este circuito debe realizar queda especificada como sigue:

si c = 0, S = (E + 1)mod8

si c = 1, S = (E – 1)mod8

Solución*: el sistema digital que se trata de describir incrementa (con c = 0) o decrementa (con c = 1) en base 8; por tanto, cuando incrementa, después del 7 va el 0, y cuando decrementa, después del 0 va el 7; el resto de las salidas son obvias, y se dan en la tabla de la Figura 5.2e.*



(a) *(b)*

	$x_1 x_2 x_3$	$z_1 z_2$
0	000	0 0
1	001	0 1
2	010	1 1
3	011	0 0
4	100	1 1
5	101	0 1
6	110	0 0
7	111	0 0

(c)

	x	y	z	u	p
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	1
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	0
10	1	0	1	0	0
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	1
15	1	1	1	1	0

(d)

	c	x	y	z	u	v	w
0	0	0	0	0	0	0	1
1	0	0	0	1	0	1	0
2	0	0	1	0	0	1	1
3	0	0	1	1	1	0	0
4	0	1	0	0	1	0	1
5	0	1	0	1	1	1	0
6	0	1	1	0	1	1	1
7	0	1	1	1	0	0	0
8	1	0	0	0	1	1	1
9	1	0	0	1	0	0	0
10	1	0	1	0	0	0	1
11	1	0	1	1	0	1	0
12	1	1	0	0	0	1	1
13	1	1	0	1	1	0	0
14	1	1	1	0	1	0	1
15	1	1	1	1	1	1	0

(e)

Figura 5.2. *(a) Representación de un sistema digital genérico; (b) ejemplo de sistema digital; (c) tabla descriptiva del funcionamiento; (d) tabla verdad correspondiente al Ejemplo 5.1; (e) tabla verdad correspondiente al Ejemplo 5.2.*

5.2 Álgebra de Boole

Para realizar de forma sistemática y cómoda tanto el análisis como la síntesis de los sistemas digitales binarios (que con frecuencia denominaremos simplemente sistemas digitales o circuitos

digitales) se utiliza como eficaz herramienta matemática el álgebra de Boole, que se puede definir mediante el conjunto de postulados que se enuncian en la siguiente Sección 5.2.1.

El álgebra de Boole fue introducida por el filósofo y matemático inglés George Boole (1815-1864) para analizar los razonamientos lógicos. Posteriormente se encontró que esta herramienta matemática era muy útil tanto en el análisis como en la síntesis de sistemas digitales.

5.2.1 Postulados del álgebra de Boole

Cualquiera álgebra se define dando un conjunto de elementos, unas operaciones binarias o leyes de composición interna que se aplican a los elementos del conjunto, y unos principios básicos o axiomas que se aplican a estas leyes de composición interna y a los elementos del conjunto. Los axiomas o postulados del álgebra de Boole que aquí se van a utilizar son los siete siguientes:

I. *Conjunto de elementos*:
Existe un conjunto de elementos, B, en el que al menos hay dos elementos diferentes:

$$\exists B \mid x, y \in B, x \neq y$$

II. *Leyes de composición interna*:
En B se definen dos leyes de composición interna, que se representan de la forma · y +, y se denominan Y y O, respectivamente:

$$\forall x, y \in B, x \cdot y \in B ; x + y \in B$$

III. *Elementos neutros únicos*:

(a) $\exists 0 ! \in B \mid \forall x \in B, x + 0 = 0 + x = x$

(b) $\exists 1 ! \in B \mid \forall x \in B, x \cdot 1 = 1 \cdot x = x$

(c) $0 \neq 1$

IV. *Conmutatividad de las leyes de composición interna*:

$$\forall x, y \in B, x + y = y + x; x \cdot y = y \cdot x$$

V. *Distributividad de las leyes de composición interna*:

$$\forall x, y, z \in B, x + (y \cdot z) = (x + y) \cdot (x + z); x \cdot (y + z) = (x \cdot y) + (x \cdot z)$$

VI. *Asociatividad de las leyes de composición interna*:

$$\forall x, y, z \in B, x + (y + z) = (x + y) + z = x + y + z; x \cdot (y \cdot z) = (x \cdot y) \cdot z = x \cdot y \cdot z$$

VII. *Elemento opuesto (complemento) único*:

$$\forall x \in B, \exists \overline{x} ! \in B \mid x + \overline{x} = 1; x \cdot \overline{x} = 0$$

Hay otros posibles conjuntos de axiomas para introducir el álgebra de Boole; aquí hemos preferido utilizar el más sencillo.

Para cada postulado se han dado dos enunciados duales, entendiendo por enunciados duales los que se pueden obtener uno a partir del otro sustituyendo "0" por "1", "+" por "·", y viceversa, en todos los lugares en que aparezcan. Por ejemplo, el primer enunciado del postulado V se expresa como

$$x + (y \cdot z) = (x + y) \cdot (x + z);$$

sustituyendo "+" por "·" en todos los lugares en los que aparecen, resulta

$$x \cdot (y + z) = (x \cdot y) + (x \cdot z),$$

que es el segundo enunciado de este postulado. De aquí se sigue que al demostrar una identidad cualquiera, simultáneamente se está demostrando su identidad dual (es lo que se

denomina **principio de dualidad**), pues para pasar de una demostración a otra basta sustituir cada postulado o teorema que se utilice por su dual. Por ejemplo, si con un razonamiento adecuado se demuestra la identidad

$$(x + y) \cdot (x + z) \cdot (y + z) = (x + y) \cdot (x + z)$$

por el principio de dualidad, simultáneamente se está demostrando la identidad

$$x \cdot y + x \cdot z + y \cdot z = x \cdot y + x \cdot z$$

A continuación, al demostrar algunos teoremas, se verán ejemplos y aplicaciones del principio de dualidad.

5.2.2 Algunos teoremas fundamentales

En esta apartado se va a presentar un conjunto de teoremas, seleccionados por la aplicación que tienen o por los hechos fundamentales que establecen; para los primeros de la relación se va a dar una demostración completa; los restantes simplemente se enunciarán. Para la demostración de los diferentes Teoremas se utilizarán los Postulados o los Teoremas previamente demostrados.

Teorema 1 $\forall\, a \in B,\ a + a = a;\ a \cdot a = a$

Este teorema se conoce también como ley de absorción o ley idempotente y, como se ve, tiene dos enunciados duales.

Demostración:

$a + a = (a + a) \cdot 1$	(Postulado III)
$= (a + a) \cdot (a + \overline{a})$	(P. VII)
$= a + (a \cdot \overline{a})$	(P. V)
$= a + 0$	(P. VII)
$= a$	(P. VIII)
	q.e.d

La demostración del enunciado dual es paralela a la anterior, utilizando los enunciados duales de los mismos postulados. En efecto:

Demostración:

$a \cdot a = a \cdot a + 0$	(P. III)
$= a \cdot a + a \cdot \overline{a}$	(P. VII)
$= a \cdot (a + \overline{a})$	(P. V)
$= a \cdot 1$	(P. VII)
$= a$	(P. III)
	q.e.d.

Los dos teoremas siguientes se refieren a los elementos neutros y completan los postulados en lo que respecta a la forma de operar con la suma y con el producto, y a la relación que hay entre ambos elementos neutros.

Teorema 2 $\forall\, a \in B,\ a + 1 = 1,\ a \cdot 0 = 0$

Demostración:

$a + 1 = (a + 1) \cdot 1$	(P. III)
$=(a + 1) \cdot (a + \overline{a})$	(P. VIII)
$=a + (1\, \overline{a})$	(P. V)

$= a + \overline{a}$ (P. III)

$= 1$ (P. VIII)

q.e.d.

Por dualidad resulta a·0=0

Teorema 3 $\overline{1} = 0,\ \overline{0} = 1$

Demostración:

$\overline{1} = \overline{1} \cdot 1$ (P. III)

$= 0$ (P. VII)

q.e.d

Por dualidad resulta $\overline{0} = 1$

Los restantes teoremas sólo van a ser enunciados.

Teorema 4 $\forall a,b \in B,\ \overline{a+b} = \overline{a} \cdot \overline{b};\ \overline{a \cdot b} = \overline{a} + \overline{b}$

Este teorema se conoce como **ley de De Morgan** y se aplica con frecuencia para obtener la función complementaria de una dada. Esta ley de De Morgan se generaliza inmediatamente y se enuncia de una forma breve diciendo que *el complemento de la suma es el producto de los complementos* y que *el complemento del producto es la suma de los complementos*.

Teorema 5 $\forall\ a,\ b \in B,\ a + a \cdot b = a;\ a \cdot (a + b) = a$

Esta es una nueva ley de absorción y, como veremos en la Sección 5.3.1, tiene evidentes aplicaciones prácticas en cuanto a la simplificación de circuitos.

Teorema 6 $\forall\ a \in B,\ \overline{\overline{a}} = a$

Según este teorema, el complemento del complemento de cualquier elemento es el propio elemento; es decir, complementaciones sucesivas de la misma variable se anulan mutuamente.

Teorema 7 $\forall\ a,\ b \in B,\ a + \overline{a} \cdot b = a + b;\ a \cdot (\overline{a} + b) = a \cdot b$

Esta es otra nueva ley de absorción.

En la Tabla 5.1 se resumen los postulados y teoremas anteriores.

Tabla 5.1. *Postulados y teoremas del álgebra de Boole.*

PI	$\exists B \mid x,\ y \in B,\quad x \neq y$	
PII	$x + y \in B\quad \forall x,\ y \in B$	$x \cdot y \in B,\quad \forall x,\ y \in B$
PIII	$x + 0 = 0 + x = x$	$x \cdot 1 = 1 \cdot x = x$
PIV	$x + y = y + x$	$x \cdot y = y \cdot x$
PV	$x + (y \cdot z) = (x+y) \cdot (x+z)$	$x \cdot (y+z) = x \cdot y + x \cdot z$
PVI	$x + (y+z) = (x+y) + z = x + y + z$	$x \cdot (y+z) = (x \cdot y) \cdot z = x \cdot y \cdot z$
PVII	$x + \overline{x} = 1$	$x \cdot \overline{x} = 0$
T1	$a + a = a$	$a \cdot a = a$
T2	$a + 1 = 1$	$a \cdot 0 = 0$
T3	$\overline{1} = 0$	$\overline{0} = 1$
T4	$\overline{a+b} = \overline{a} \cdot \overline{b}$	$\overline{a \cdot b} = \overline{a} + \overline{b}$
T5	$a + ab = a$	$a \cdot (a + b) = a$
T6	$(\overline{\overline{a}}) = a$	$(\overline{\overline{a}}) = a$
T7	$a + \overline{a}b = a + b$	$a \cdot (\overline{a} + b) = a \cdot b$

EJEMPLO 5.3

Utilizando los postulados y teoremas del álgebra de Boole, verificar que se cumple la siguiente igualdad:

$$A = (x + \overline{y} + xy)(x + \overline{y})\,\overline{x}y = 0$$

Solución: *por el teorema 5 es* $x + xy = x$.

Por tanto, $x + \overline{y} + xy = x + xy + \overline{y} = x + \overline{y}$

Por el teorema 1 es $(x + \overline{y})(x + \overline{y}) = x + \overline{y}$, *quedando* $A = (x + \overline{y})\,\overline{x}y$.

Por el postulado V es $A = (x + \overline{y})\,\overline{x}y = (x\overline{x}y) + (\overline{y}\,\overline{x}y)$.

Finalmente, aplicando el postulado VII y el teorema 2 queda

$$A = (x\overline{x}y) + (\overline{y}\,\overline{x}y) = (x\overline{x}y) + (y\overline{y}\,\overline{x}) = (0 \cdot y) + (0 \cdot \overline{x}) = 0 + 0 = 0$$

Hasta este punto el álgebra de Boole no pasa de ser una construcción más o menos elaborada, pero no está demostrado siquiera que exista un ejemplo concreto de tal estructura matemática, pues los postulados podrían ser contradictorios. Por ello es aconsejable presentar ahora algún ejemplo concreto de álgebra de Boole.

El ejemplo más sencillo de álgebra de Boole es aquel cuyo conjunto de elementos es B={0,1}. En este caso las operaciones +, ·, y ¯ (complementación) se obtienen por aplicación de los postulados y teoremas anteriores, quedando tal como se dan en la Figura 5.3.

+	0	1
0	0	1
1	1	1

·	0	1
0	0	0
1	0	1

–	
0	1
1	0

Figura 5.3. *Operaciones en el álgebra de Boole* B_2.

Es fácil comprobar que con estas operaciones se cumplen todos los postulados anteriores. Este álgebra de Boole se designa como B_2.

Otro ejemplo muy conocido de álgebra de Boole es el álgebra de conjuntos. En este caso el conjunto B está formado por todos los subconjuntos del conjunto universal, las dos leyes de composición interna son la intersección de conjuntos (·) y la unión de conjuntos (+); los elementos neutros son el conjunto universal (1) y el conjunto nulo (0); como complemento de un elemento *A* se define el conjunto complemento de *A, C(A)*. Si el conjunto universal tiene un número infinito de elementos, el conjunto de elementos B tiene un número infinito de elementos. En general, cuando el conjunto B tiene un número de elementos finito, *k*, se puede demostrar que este número de elementos del álgebra de Boole finita correspondiente es 2^k.

5.3 Funciones de conmutación

Las entradas x_1, x_2, y x_3, y las salidas z_1 y z_2 del sistema digital de la Figura 5.2b son variables de conmutación. Definimos como **variable de conmutación** a un símbolo cualquiera, normalmente una letra, que en un instante determinado sólo puede tomar uno de los valores 0 ó 1.

Dado el conjunto C={0,1}, una **función de conmutación** de *n* variables, $f(x_1, ..., x_n)$, es una aplicación del conjunto producto cartesiano $\{0,1\}^n$ en el conjunto {0,1}.

EJEMPLO 5.4

El conjunto $\{0,1\}^2$ es {00,01,10,11}. Una función de conmutación de dos variables puede ser {00,01,10,11} → {0,1,1,0}. Es decir, a la combinación de entradas 00 le corresponde 0; a la entrada 01 le corresponde 1 a la salida; a 10 se le asigna 1 de salida; 11 produce 0.

El conjunto $\{0,1\}^3$ es {000, 001, 010, 011, 100, 101, 110, 111}. Esta forma de dar la función es totalmente equivalente a la tabla verdad.

Es inmediato comprobar que $\{0,1\}^n$ tiene 2^n elementos. Dado que {0,1} tiene dos elementos, es también inmediato comprobar que hay 2^{2^n} funciones de conmutación de n variables.

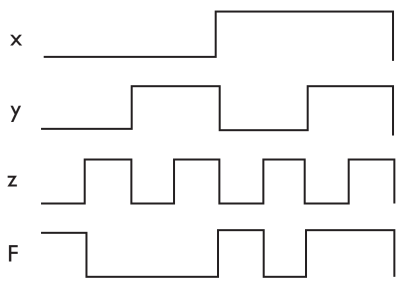
Una función de conmutación se puede dar de varias formas; las más usuales son las siguientes:

a) En forma de tabla verdad, tal como se ha visto en la Figura 5.2.

b) Gráficamente, mediante un **cronograma**, tal como se hace en la Figura 5.4a para una función *F* de las variables *x, y, z.*

c) Mediante una igualdad, como la siguiente:

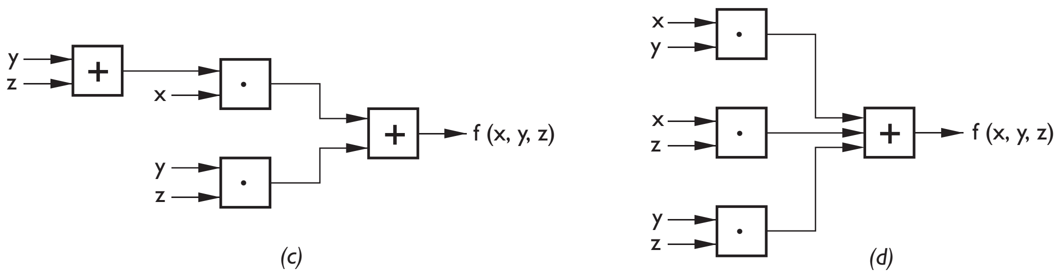
$$f(x,y,z) = x \cdot (y + z) + y \cdot z$$



(a)

xyz	F
000	0
001	0
010	0
011	1
100	0
101	1
110	1
111	1

(b)



(c) (d)

Figura 5.4. *(a) Cronograma de una función F(x, y, z); (b)tabla verdad de f(x,y,z); (c) circuito sintetizador de f(x,y,z); (d) otro circuito sintetizador.*

La expresión de la forma c) significa que, para cada combinación de valores de las entradas, el valor 0 ó 1 que $f(x,y,z)$ asigna se determina evaluando la expresión a la derecha del signo de igualdad. Concretamente esta función es {000, 001, 010, 011, 100, 101, 110, 111} → {0, 0, 0, 1, 0, 1, 1, 1}. En efecto, para $xyz = 000$ es $x \cdot (y + z) + y \cdot z = 0 \cdot (0 + 0) + 0 \cdot 0 = 0 \cdot 0 + 0 \cdot 0 = 0 + 0 = 0$; para $xyz = 001$ es $x \cdot (y + z) + y \cdot z = 0 \cdot (0 + 1) + 0 \cdot 1 = 0 \cdot 1 + 0 \cdot 1 = 0 + 0 = 0$; y así sucesivamente para las restantes combinaciones de entradas.

En la Figura 5.4b se tiene la expresión de esta función en forma de tabla verdad. Si se dispone de dispositivos que sinteticen las funciones Y y O, un posible circuito para sintetizar $f(x,y,z)$ es de la Figura 5.4c; este circuito tiene tres niveles de puertas: un primer nivel con una puerta **O** con dos entradas externas (x,y), un segundo nivel con dos puertas **Y**, y un tercer nivel con una puerta **O** que produce la salida $f(x,y,z)$. Este circuito se obtiene trasladando directamente la expresión algebraica de partida. Aplicando a esta expresión la propiedad distributiva resulta:

$$f(x,y,z) = x \cdot (y + z) + y \cdot z = x \cdot y + x \cdot z + y \cdot z$$

Trasladando directamente esta última expresión resulta el circuito de la Figura 5.4d, que tiene un primer nivel con tres puertas **Y**, y un nivel de salida con una puerta **O**. Es decir, una misma función se puede sintetizar con diferentes circuitos.

EJEMPLO 5.5

En una habitación hay un foco de luz que deseamos poder controlar independientemente desde tres puntos diferentes. Diseñar un sistema digital que realice esta función.

***Solución**: la primera etapa en el diseño de este sistema digital va a ser obtener la función de conmutación que se ha de sintetizar. Supongamos que en tres puntos diferentes se tienen sendos conmutadores con los que se quiere poder encender o apagar la luz. Los conmutadores se van a representar mediante las variables de conmutación A, B, C, a las que convenimos en asignarles el valor "0" cuando el correspondiente conmutador esté abierto, y el valor "1" cuando esté cerrado (esto no es más que un convenio, y podría utilizarse perfectamente el convenio contrario). El sistema digital que se diseñe tendrá como entradas A, B, C, y producirá salida "1" cuando la luz haya de estar encendida, y salida 0 cuando deba estar apagada. Si decidimos, como parece lógico, que la luz esté apagada (f = 0) cuando los tres conmutadores estén abiertos (A = 0, B = 0, C = 0), la primera fila de la tabla verdad será 000 → 0. Cada cambio en una de las entradas A, B, C, hará que cambie el estado de la luz (de encendida a apagada, y viceversa). De lo anterior se concluye inmediatamente que la tabla verdad para este sistema digital es la de la Figura 5.5a. Es decir, f ha de valer 1 cuando ("A es cero" **Y** "B es cero" **Y** "C es 1") **O** ("A es cero" **Y** "B es 1" **Y** "C es cero") **O** ("A es 1" **Y** "B es cero" **Y** "C es cero") **O** ("A es 1" **Y** "B es 1" **Y** "C es 1"). Utilizando también las variables complementadas* $\overline{A}$*,* $\overline{B}$ *y* $\overline{C}$*, este enunciado se podría expresar como sigue:*

***f** ha de valer **1** cuando*
("$\overline{A}$ *es 1" **Y** "*$\overline{B}$ *es 1" **Y** "C es 1") **O** ("*$\overline{A}$ *es 1" **Y** "B es 1" **Y** "*$\overline{C}$ *es 1") **O***
*("A es 1" **Y** "*$\overline{B}$ *es 1" **Y** "*$\overline{C}$ *es 1") **O** ("A es 1" **Y** "B es 1" **Y** "C es 1")*

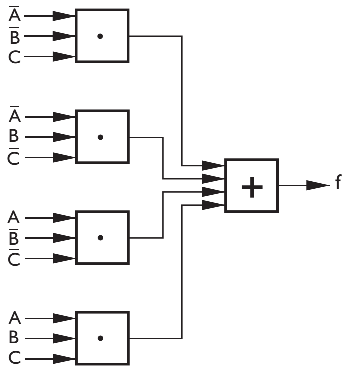
Este enunciado nos lleva a la siguiente expresión algebraica:

$$f = \overline{A} \cdot \overline{B} \cdot C + \overline{A} \cdot B \cdot \overline{C} + A \cdot \overline{B} \cdot \overline{C} + A \cdot B \cdot C$$

*Y a partir de esta expresión algebraica se obtiene, utilizando puertas **O** e **Y**, el circuito de la Figura 5.5b. Es decir, resulta un circuito con dos niveles; en el primer nivel hay una puerta **Y** por cada 1 de f en la tabla verdad; el segundo nivel consta de una puerta **O** de salida.*

ABC	f
000	0
001	1
010	1
011	0
100	1
101	0
110	0
111	1

(a)



(b)

Figura 5.5. *(a) Tabla verdad correspondiente al Ejemplo 5.5; (b) sistema digital que se obtiene de la tabla verdad anterior.*

5.3.1 Álgebra de conmutación

El conjunto F de todas las funciones de conmutación de *n* variables, con la relación de igualdad y las operaciones Y, O, y complementación definidas a continuación, forma un álgebra de Boole finita, que se conoce como **álgebra de conmutación**.

Relación de igualdad: dos funciones de conmutación de *n* variables son iguales si para todos los posibles valores de las variables se establecen las mismas asignaciones. Por ejemplo, es inmediato comprobar que son iguales las funciones $f(x,y,z)$ dada en la tabla verdad de la Figura 5.4b, y $g(x,y,z) = x \cdot y + (x + y) \cdot z$.

Y	0	1
0	0	0
1	0	1

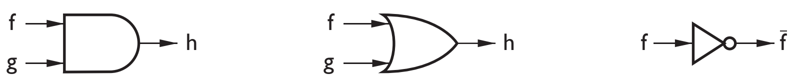
(a)

O	0	1
0	0	1
1	1	1

(c)

–	
0	1
1	0

(e)



(b) *(d)* *(f)*

Figura 5.6. *(a) Tabla de la operación* **Y***; (b) circuito* **Y***; (c) tabla de la operación* **O***; (d) circuito* **O***; (e) complementación; (f) complementador.*

Operación Y (·): dadas $f, g \in F$, se define $h = f \cdot g$, $h \in F$, como sigue: "para cada elemento de $\{0,1\}^n$, h asigna el valor 1 si y sólo si *f* y *g* asignan ambas el valor 1; en cualquier otro caso *h* asigna el valor 0". De acuerdo con esta definición, la operación Y queda descrita mediante la tabla de la Figura 5.6a, que coincide con la dada en la Figura 5.3 para la misma operación Y. En la Figura 5.6b se da el símbolo que se va a utilizar para cualquier sistema digital binario (electrónico, por ejemplo) que realice la operación Y.

Operación O (+): dadas $f, g \in$ F, se define $h = f + g$, $h \in$ F, como sigue: "para cada elemento de $\{0,1\}^n$, h asigna el valor 0 si y sólo si f y g asignan el valor 0; en cualquier otro caso h asigna el valor 1". De acuerdo con esta definición, la operación O queda descrita mediante la tabla de la Figura 5.6c, que coincide con la dada en la Figura 5.3 para la misma operación O. En la Figura 5.6d se tiene el símbolo para cualquier sistema digital que realice la operación O.

Complemento de una función: dada $f \in$ F, se define $\overline{f} \in$ F como sigue: "para cada elemento de $\{0,1\}^n$, $\overline{f}$ asigna el valor 0 si y sólo si f asigna el valor 1, y viceversa". La operación de complementación queda descrita mediante la tabla de la Figura 5.6e, y en la Figura 5.6f se tiene el símbolo de un complementador.

Elemento cero: la función cero (0^n) es la que a cada elemento de $\{0,1\}^n$ le asigna el valor 0.

Elemento uno: la función uno (1^n) es la que a cada elemento de $\{0,1\}^n$ le asigna el valor 1.

5.3.2 Minterms y maxterms

A continuación vamos a definir dos tipos particulares de funciones de n variables, los minterms o términos mínimos y los maxterms o términos máximos.

Funciones minterm: dado un número entero I, $0 \leq I \leq 2^n - 1$, y su expresión en base dos con n bits, $(i_1 \ldots i_n)$, se define la función minterm $m_I(x_1,\ldots,x_n)$ de la forma:

$$m_I(x_1,\ldots,x_n)=1 \qquad \text{si } x_1=i_1,\ldots,x_n = i_n$$
$$m_I(x_1,\ldots,x_n)=0 \qquad \text{en cualquier otro caso}$$

Es claro que esta función asigna el valor 1 para una sola n-tupla (de aquí la denominación de término mínimo) de $\{0,1\}^n$ y esta n-tupla representa, por tanto, unívocamente al minterm. Para n variables hay 2^n minterms. En la Figura 5.7a se dan algunos minterms para el caso de tres variables.

Un minterm se puede expresar como un producto de la forma

$$x_1^{i_1} \ldots x_n^{i_n}$$

donde

$$x_j^{i_j} = x_j \qquad \text{si } i_j = 1$$
$$x_j^{i_j} = \overline{x}_j^{i_j} \qquad \text{si } i_j = 0$$

Por ejemplo, para tres variables, x, y, z, es

$$m_3 = \overline{x}yz, \qquad m_4 = x\overline{y}\,\overline{z}, \qquad m_6 = xy\overline{z}$$

En la Figura 5.7b se relacionan los ocho minterms de tres variables, expresados como productos Y. En cada fila se da el producto Y que corresponde al minterm con el 1 en esa fila de la tabla verdad. Como puede observarse, cada variable aparece en el producto complementada o sin complementar, dependiendo de que esta variable en esa fila tome el valor 0 ó 1, respectivamente.

Funciones maxterm: dado un número entero I, $0 \leq I \leq 2^n - 1$, y su expresión en base dos con n bits, $(i_1 \ldots i_n)$, se define la función maxterm $M_I(x_1, \ldots x_n)$ de la forma:

$$M_I(x_1,\ldots,x_n) = 0 \qquad \text{si } x_1 = i_1,\ldots, x_n = i_n$$
$$M_I(x_1,\ldots,x_n) = 1 \qquad \text{en cualquier otro caso}$$

Es claro que esta función asigna el valor 0 para una sola n-tupla (es decir, asigna el valor 1 a todas las 2^n diferentes n-tuplas menos a una, y de aquí la denominación de término máximo) de $\{0,1\}^n$; esta n-tupla representa, por tanto, unívocamente al maxterm. Para n variables hay 2^n maxterms. En la Figura 5.7a se dan algunos maxterms para el caso de tres variables.

Un maxterm se puede expresar como una suma de la forma

$$\overline{x}_1^{i_1} + \ldots + \overline{x}_n^{i_n}$$

donde

$$\overline{x}_j^{i_j} = \overline{x}_j \qquad \text{si } i_j = 1$$

$$\overline{x}_j^{i_j} = x_j \qquad \text{si } i_j = 0$$

Por ejemplo, para tres variables, x, y, z, es

$$M_0 = x + y + z,\ M_2 = x + \overline{y} + z,\ M_5 = \overline{x} + y + \overline{z}$$

En la Figura 5.7b se dan los ocho maxterms de tres variables, expresados en forma de suma O. En cada fila se da la suma O que corresponde al maxterm con el 0 (cero) en esa fila de la tabla verdad. Como puede observarse, cada variable aparece en la suma complementada o sin complementar, dependiendo de que esta variable en esa fila tome el valor 1 ó 0, respectivamente (nótese que el criterio cambia con respecto al enunciado antes para los minterms).

	$x_1 x_2 x_3$	m_0 m_3 m_4 m_6	…	M_0 M_2 M_5 M_7	f_1 f_2
0	000	1 0 0 0		0 1 1 1	0 0
1	001	0 0 0 0		1 1 1 1	1 0
2	010	0 0 0 0		1 0 1 1	1 0
3	011	0 1 0 0		1 1 1 1	0 1
4	100	0 0 1 0		1 1 1 1	0 1
5	101	0 0 0 0		1 1 0 1	0 1
6	110	0 0 0 1		1 1 1 1	0 1
7	111	0 0 0 0		1 1 1 0	1 0

(a)

	xyz	minterms	Maxterms
0	000	$\overline{x}\,\overline{y}\,\overline{z}$	$x+y+z$
1	001	$\overline{x}\,\overline{y}\,z$	$x+y+\overline{z}$
2	010	$\overline{x}\,y\,\overline{z}$	$x+\overline{y}+z$
3	011	$\overline{x}\,y\,z$	$x+\overline{y}+\overline{z}$
4	100	$x\,\overline{y}\,\overline{z}$	$\overline{x}+y+z$
5	101	$x\,\overline{y}\,z$	$\overline{x}+y+\overline{z}$
6	110	$x\,y\,\overline{z}$	$\overline{x}+\overline{y}+z$
7	111	$x\,y\,z$	$\overline{x}+\overline{y}+\overline{z}$

(b)

Figura 5.7. *(a) Funciones minterms y maxterms de tres variables; (b) relación de los minterms y maxterms de tres variables.*

La utilidad de las funciones minterm y maxterm reside en que cualquier función de conmutación se puede expresar en términos de estas funciones, según indica el siguiente teorema de Shannon.

Teorema de Shannon (primera forma): *toda función de conmutación de n variables, $f(x_1,...,x_n)$, puede expresarse como una suma única de minterms.*

Demostración: $f(x_1,...,x_n)$ se puede desarrollar con respecto a x_1 de la forma siguiente: [5.1]

$$f(x_1,...,x_n) = \overline{x}_1 f(0, x_2,...,x_n) + x_1 f(1, x_2,...,x_n)$$

donde $f(0,x_2,...,x_n)$ y $f(1,x_2,...,x_n)$ son funciones de n–1 variables que coinciden con $f(x_1,...,x_n)$ cuando $x_1 = 0$ o $x_1 = 1$, respectivamente.

En efecto, si $x_1 = 0$, es $\overline{x}_1 = 1$, y [5.1] queda como sigue:

$$f(x_1,x_2,....,x_n) = 1 \cdot f(0,x_2,...,x_n) + 0 \cdot f(1,x_2,...,x_n) = f(0,x_2,...,x_n)$$

Si es x_1=1, entonces $\overline{x}_1 = 0$, con lo que de [5.1] resulta

$$f(x_1,x_2,....,x_n) = 0 \cdot f(0,x_2,...,x_n) + 1 \cdot f(1,x_2,..., x_n) = f(1,x_2,...,x_n)$$

Aplicando este mismo desarrollo a las restantes variables queda:

$$\begin{aligned} f(x_1,...,x_n) &= \overline{x}_1 f(0,x_2,...,x_n) + x_1 f(1,x_2,...,x_n) = \\ &= \overline{x}_1 \overline{x}_2 f(0,0,x_3,...,x_n) + \overline{x}_1 x_2 f(0,1,x_3, ... ,x_n) + \\ &+ x_1 \overline{x}_2 f(1,0,x_3,...x_n) + x_1 x_2 f(1,1,x_3,...,x_n) = \\ &= ... = \overline{x}_1 \overline{x}_2 ... \overline{x}_n \cdot f(0,0,...,0) + ... + x_1 x_2 ... x_n \cdot f(1,1,...,1) \end{aligned}$$

Resulta una suma de los 2^n minterms, cada uno de los cuales está multiplicado (función Y) por una constante, $f(i_1,...i_n)$, que es el valor que la función aplica a la n-tupla correspondiente. Por tanto, de los 2^n minterms, sólo permanecerán aquellos que estén multiplicados por 1, es decir, aquellos para los que la función f asigna el valor 1. En otras palabras, en la expresión de una función como suma de minterms, hay un minterm por cada 1 que aparezca en la tabla verdad de la función. Por ejemplo, las funciones f_1 y f_2 de la Figura 5.7a tienen la siguiente expresión en forma de suma de minterms:

$$f_1 = \overline{x}\,\overline{y}z + \overline{x}y\overline{z} + xyz$$

$$f_2 = \overline{x}yz + x\overline{y}\,\overline{z} + x\overline{y}z + xy\overline{z}$$

En la Figura 5.8a y b se dan las realizaciones físicas de f_1 y f_2 en dos niveles de puertas utilizando circuitos Y y O.

Regla práctica: *dada una función de conmutación expresada en forma de tabla verdad, para obtener su expresión en forma de suma de minterms hay que escribir un producto Y (un minterm) por cada 1 que aparezca en la columna del valor de la función en la tabla verdad, siendo ese producto Y el que en la Figura 5.7b ocupa la misma posición que el 1.* Esto es lo que se ha hecho para obtener las expresiones de f_1 y f_2 dadas anteriormente. En efecto, f_1 tiene tres unos (filas 1, 2 y 7), y en su expresión como suma **O** de minterms aparecen los minterms de las filas 1, 2 y 7. De forma similar se ha obtenido f_2.

De todo lo anterior resulta que, para dar una función, basta con dar los minterms que la forman. La forma de hacerlo es la que se utiliza a continuación para las mismas funciones anteriores f_1 y f_2:

$$f_1 = \Sigma m(1,2,7)$$

$$f_2 = \Sigma m(3,4,5,6)$$

EJEMPLO 5.6

Obtener la expresión en forma de suma de minterms de la función

$$F(x,y,z) = x\,(\overline{y} + z) + yz$$

Solución: *desarrollando la función anterior y ampliando los productos con las variables ausentes queda:*

$$F(x,y,z) = x\,(\overline{y} + z) + yz = x\overline{y} + xz + yz = x\overline{y}\,(z + \overline{z}) + x(y + \overline{y})z + (x + \overline{x})yz$$
$$= x\overline{y}z + x\overline{y}\,\overline{z} + xyz + x\overline{y}z + xyz + \overline{x}yz = x\overline{y}z + x\overline{y}\,\overline{z} + xyz + \overline{x}yz$$

Es decir, $F(x,y,z) = \Sigma m(3,4,5,7)$

Hay un desarrollo similar para el producto de maxterms, que viene expresado por la segunda forma del teorema de Shannon.

Teorema de Shannon (segunda forma): *toda función de conmutación de n variables,* $f(x_1,...,x_n)$*, puede expresarse como un producto único de maxterms.*

La demostración es paralela a la de la primera forma de este teorema (por ello no se va a dar, aunque se recomienda al lector que, como ejercicio, trate de construirla), pero partiendo ahora del desarrollo de $f(x_1,...x_n)$ con respecto a x_1 de la forma siguiente:

$$f(x_1,...,x_n) = [x_1 + f(0,x_2,...,x_n)] \cdot [\overline{x}_1 + f(1,x_2,...,x_n)]$$

Aplicando esta segunda forma del teorema de Shannon resulta un producto de maxterms, tal que hay un maxterm por cada 0 que aparezca en la tabla verdad de la función. Por ejemplo, las funciones f_1 y f_2 de la Figura 5.7 tienen la siguiente expresión en forma de producto de maxterms:

$$f_1 = (x+y+z)\cdot(x+\overline{y}+\overline{z})\cdot(\overline{x}+y+z)\cdot(\overline{x}+y+\overline{z})\cdot(\overline{x}+\overline{y}+z)$$

$$f_2 = (x+y+z)\cdot(x+y+\overline{z})\cdot(x+\overline{y}+z)\cdot(\overline{x}+\overline{y}+\overline{z})$$

En la Figura 5.8c y d se dan las realizaciones físicas de f_1 y f_2 utilizando circuitos O a Y en dos niveles de puertas.

Regla práctica: *dada una función de conmutación expresada en forma de tabla verdad, para obtener su expresión en forma de producto Y de maxterms hay que escribir una suma O (un maxterm) por cada 0 (cero) que aparezca en la columna del valor de la función en la tabla verdad, siendo esa suma O la que en la Figura 5.7b ocupa la misma posición que el 0.* Esto es lo que se ha hecho para obtener las expresiones de f_1 y f_2 dadas anteriormente. En efecto, f_1 tiene cinco ceros (filas 0, 3, 4, 5 y 6), y en su expresión como suma **O** de maxterms aparecen los maxterms de las filas 0, 3, 4, 5 y 6. De forma similar se ha obtenido f_2.

De todo lo anterior resulta que, para dar una función, basta con dar los maxterms que la forman. La forma de hacerlo es la que se utiliza a continuación para las mismas funciones anteriores f_1 y f_2:

$$f_1 = \Pi M(0,3,4,5,6)$$
$$f_2 = \Pi M(0,1,2,7)$$

EJEMPLO 5.7

Obtener la expresión en forma de suma de maxterms de la función

$$F(x,y,z) = x \cdot (\overline{y} + z) + yz$$

Solución: *desarrollando la función anterior y ampliando las sumas con las variables ausentes queda:*

$$F(x,y,z) = x \cdot (\overline{y}+z) + yz = (x+y)\cdot(x+z)\cdot(\overline{y}+z+y)\cdot(\overline{y}+z+z) = \{(x+y)(z+\overline{z})\}\{(x+z)(y+\overline{y})\}\{(x+\overline{x})(\overline{y}+z)\} =$$
$$= (x+y+z)(x+y+\overline{z})(x+y+z)(x+\overline{y}+z)(x+\overline{y}+z)(\overline{x}+\overline{y}+z) = (x+y+z)(x+y+\overline{z})(x+\overline{y}+z)(\overline{x}+\overline{y}+z)$$

Es decir, $F(x,y,z) = \Pi M(0,1,2,6)$

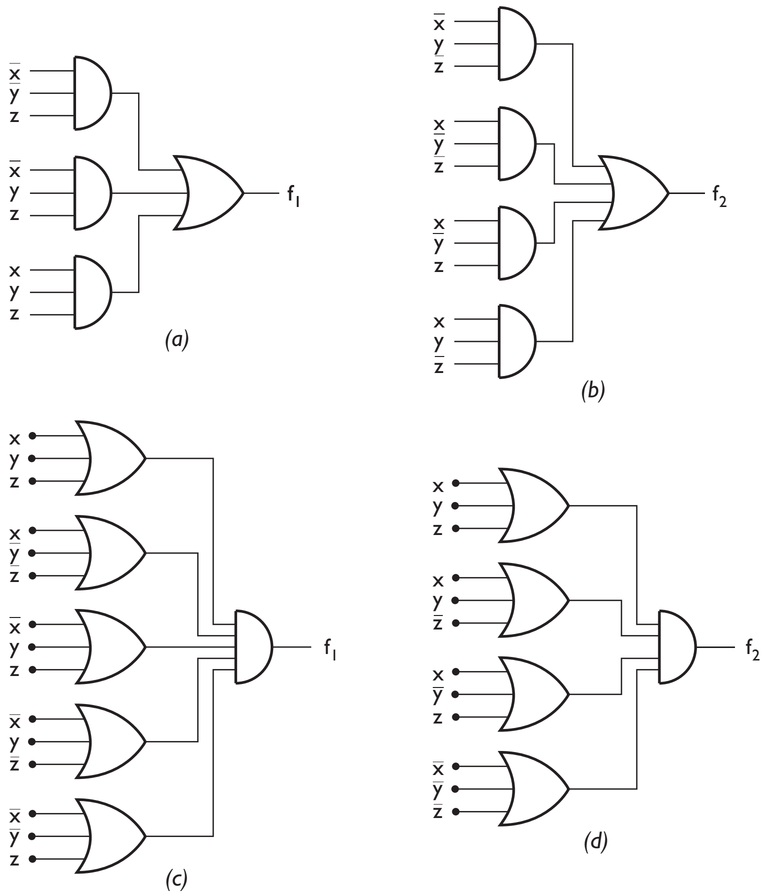


Figura 5.8. *(a) y (b) Realización Y a O de f_1 y f_2; (c) y (d) realización O a Y de f_1 y f_2.*

Se tienen, por tanto, dos alternativas para realizar cualquier función de conmutación con estructuras de dos niveles: bien con un nivel de puertas Y que alimentan a una puerta O de salida, bien con un nivel de puertas O que alimentan a una puerta Y de salida. Ninguna de estas dos formas de realización es a priori ventajosa frente a la otra, y los procedimientos que sean aplicables a una de ellas se pueden aplicar de forma paralela a la otra. Por esta razón, en todo lo que sigue nos referiremos únicamente a la realización en forma de suma de productos, pero siempre se ha de tener presente que hay un desarrollo paralelo correspondiente al producto de sumas.

EJEMPLO 5.8

Las acciones de una sociedad están distribuidas en cuatro lotes con los siguientes porcentajes: A, 20%; B, 22%; C, 27%; D, 31%. Los acuerdos en la sociedad se toman por mayoría absoluta. Diseñar un sistema digital que tenga como entradas cuatro señales A, B, C, y D, que valdrán 0 ó 1 según que el correspondiente accionista vote en contra o a favor de una propuesta; el sistema digital deberá producir salida 1 cuando se alcance mayoría absoluta en una propuesta.

Solución: *es inmediato comprobar que la tabla verdad correspondiente a la función que describe el sistema a diseñar es la de la Figura 5.9a, y la realización Y a O de F como suma de minterms es la dada en la Figura 5.9b, que incluye ocho puertas Y, cada una de cuatro entradas, más una puerta O de ocho entradas; en total, nueve puertas. Para simplificar el dibujo, las cuatro entradas de cada puerta Y se han dibujado como una sola línea.*

La expresión algebraica de esta función como suma de minterms es

$$F = \overline{A}\overline{B}CD + \overline{A}B\overline{C}D + \overline{A}BCD + A\overline{B}\overline{C}D + A\overline{B}CD + AB\overline{C}D + ABC\overline{D} + ABCD$$

Es decir, $F = \Sigma m(3,5,7,9,11,13,14,15)$

EJEMPLO 5.9

Aplicando teoremas y postulados del álgebra de Boole, obtener para la función F del Ejemplo 5.8 otra expresión como suma de productos que incluya menos productos.

Solución: *se podrían realizar, por ejemplo, las siguientes sustituciones en F:*

$$\overline{A}\,\overline{B}CD + \overline{A}BCD + A\overline{B}CD + ABCD = \overline{A}(B+\overline{B})CD + A(B+\overline{B})CD =$$
$$= \overline{A}\cdot 1\cdot CD + A\cdot 1\cdot CD = \overline{A}\cdot CD + A\cdot CD = (\overline{A}+A)CD = 1\cdot CD = CD$$
$$\overline{A}B\overline{C}D + \overline{A}BCD + AB\overline{C}D + ABCD = \overline{A}B(\overline{C}+C)D + AB(\overline{C}+C)D =$$
$$= \overline{A}BD + ABD = (\overline{A}+A)BD = BD$$
$$A\overline{B}\overline{C}D + A\overline{B}CD + AB\overline{C}D + ABCD = A\overline{B}(\overline{C}+C)D + AB(\overline{C}+C)D =$$
$$= A\overline{B}D + ABD = A(\overline{B}+B) = AD$$
$$ABC\overline{D} + ABCD = ABC(\overline{D}+D) = ABC$$

Utilizando el teorema 1 del álgebra de Boole (a + a = a), F se puede escribir de la forma

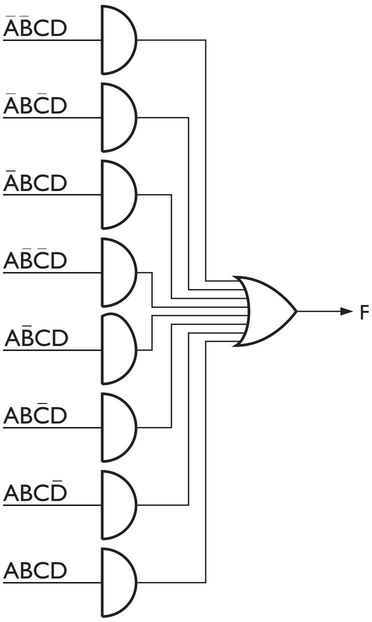
$$F = (\overline{A}\overline{B}CD + \overline{A}BCD + A\overline{B}CD + ABCD) + (\overline{A}B\overline{C}D + \overline{A}BCD + AB\overline{C}D + ABCD) +$$
$$+ (A\overline{B}\overline{C}D + A\overline{B}CD + AB\overline{C}D + ABCD) + (ABC\overline{D} + ABCD) = CD + BD + AD + ABC$$

Significa esto que para realizar F basta utilizar un circuito Y a O con cuatro puertas Y (tres de dos entradas y una de tres entradas) y una puerta O de cuatro entradas; es decir, se reduce de manera apreciable el número de puertas (cinco frente a nueve) y se reduce también el número de entradas de las puertas (trece frente a cuarenta).

Dado el interés práctico que tiene simplificar funciones (puesto de manifiesto en el ejemplo anterior), vamos a esbozar algunas posibilidades para "minimizar" las expresiones de una función de conmutación que venga dada como suma de minterms o como producto de maxterms, entendiendo por **minimizar** el reducir cuanto se pueda el número de productos o el número de sumas, y que en esos productos o sumas aparezca el menor número posible de variables. Con este objetivo se va a introducir en la siguiente sección el concepto de adyacencia.

	ABCD	Suma de votos	F
0	0000	0	0
1	0001	31	0
2	0010	27	0
3	0011	58	1
4	0100	22	0
5	0101	53	1
6	0110	49	0
7	0111	80	1
8	1000	20	0
9	1001	51	1
10	1010	47	0
11	1011	78	1
12	1100	42	0
13	1101	73	1
14	1110	69	1
15	1111	100	1

(a)



(b)

Figura 5.9. *(a) Tabla verdad del Ejemplo 5.8; (b) realización Y a 0 de esta función como suma de minterms.*

5.3.3 Adyacencias

A continuación vamos a definir de una manera recurrente las adyacencias de cualquier orden.

Definición 1: un minterm es una adyacencia de orden cero.

Definición 2: dos adyacencias de orden i, siendo i cualquier entero mayor o igual que cero, forman una adyacencia de orden $i + 1$ si las dos adyacencias de partida se expresan como productos en los que aparezcan las mismas variables, y únicamente se diferencian en que una de esas variables aparece en un producto complementada y en el otro sin complementar. La adyacencia de orden $i + 1$ se expresa como un producto en el que aparecen todas las variables idénticas en las dos adyacencias de orden i originarias (es decir no aparece la variable en que difieren).

Por ejemplo, refiriéndonos a las variables x, y, z, u, dos minterms o adyacencias de orden cero son:

$$x\,\overline{y}\,\overline{z}\,u$$

$$x\,\overline{y}\,\overline{z}\,\overline{u}$$

Estas dos adyacencias de orden cero forman una adyacencia de orden uno, dado que en ambos minterms se tienen las variables x, $\overline{y}$, $\overline{z}$ (todas menos una) idénticas. La adyacencia de orden uno resultante se expresa como un producto de estas tres variables, es decir, $x\,\overline{y}\,\overline{z}$. Concretamente se ha utilizado la siguiente sustitución:

$$x\overline{y}\,\overline{z}u + x\overline{y}\,\overline{z}\,\overline{u} = x\,\overline{y}\,\overline{z}\,(u + \overline{u}) = x\,\overline{y}\,\overline{z}\cdot 1 = x\,\overline{y}\,\overline{z}$$

Dos adyacencias de orden uno son $x\overline{y}\,\overline{z}$, $x\overline{y}z$. Estas dos adyacencias de primer orden forman una adyacencia de segundo orden expresada mediante el producto $x\overline{y}$.

Dada una adyacencia de un orden cualquiera referida a unas determinadas variables, es inmediato obtener las adyacencias de orden cero de las que procede. Por ejemplo, sean las variables x,y,z,u y sea la adyacencia de orden dos $x\overline{z}$. Las adyacencias de orden cero de las que procede (serán cuatro) se obtienen como sigue:

$$x\overline{z} = x(y+\overline{y})\overline{z}\,(u + \overline{u}) = xy\overline{z}u + xy\overline{z}\,\overline{u} + x\overline{y}\,\overline{z}u + x\overline{y}\,\overline{z}\,\overline{u}$$

El concepto de adyacencia es muy útil con vistas a la realización de las funciones de conmutación.

EJEMPLO 5.10

Obtener las adyacencias de las funciones del Ejemplo 5.2.

Solución: *estas funciones, dadas en forma de tabla verdad en la Figura 5.2e, expresadas como suma de minterms son:*

$$\mathbf{u} = \Sigma m(3,4,5,6,8,13,14,15)$$
$$\mathbf{v} = \Sigma m(1,2,5,6,8,11,12,15)$$
$$\mathbf{w} = \Sigma m(0,2,4,6,8,10,12,14)$$

Teniendo en cuenta la expresión como producto de cada minterm, para la función **u** *resultan las siguientes adyacencias de primer orden:*

$$4\text{-}5 \rightarrow \overline{c}x\overline{y},\; 5\text{-}13 \rightarrow x\overline{y}z,\; 6\text{-}14 \rightarrow xy\overline{z},\; 13\text{-}15 \rightarrow cxz,\; 14\text{-}15 \rightarrow cxy$$

No se incluyen los mintems

$$3 \rightarrow \overline{c}xyz,\; 8 \rightarrow cx\overline{y}\,\overline{z}$$

que quedan como adyacencias de orden cero. Además, las adyacencias de primer orden no generan ninguna adyacencia de segundo orden.

Para la función v resultan las siguientes adyacencias de primer orden:

$$1\text{-}5 \rightarrow \overline{c}\,\overline{y}z,\; 2\text{-}6 \rightarrow \overline{c}y\overline{z},\; 8\text{-}12 \rightarrow c\overline{y}\,\overline{z},\; 11\text{-}15 \rightarrow cyz$$

Se incluyen todos los minterms y no se obtienen adyacencias de segundo orden. También resulta que la función **v** *no depende de x.*

Para la función **w** *resultan, entre otras, las siguientes adyacencias de primer orden, en las que están incluidos todos los minterms de* **w***:*

$$0\text{-}4 \rightarrow \overline{c}\,\overline{y}\,\overline{z},\; 2\text{-}6 \rightarrow \overline{c}y\overline{z},\; 8\text{-}12 \rightarrow c\overline{y}\,\overline{z},\; 10\text{-}14 \rightarrow cy\overline{z}$$

A partir las anteriores cuatro adyacencias de primer orden se pueden construir, entre otras, las dos adyacencias de segundo orden siguientes:

$$0\text{-}2\text{-}4\text{-}6 \rightarrow \overline{c}\,\overline{z},\; 8\text{-}10\text{-}12\text{-}14 \rightarrow c\overline{z}$$

Y de estas dos adyacencias de segundo orden se obtiene la adyacencia de tercer orden

$$0\text{-}2\text{-}4\text{-}6\text{-}8\text{-}10\text{-}12\text{-}14 \rightarrow \overline{z}$$

Por tanto, la función **w** *sólo depende de la entrada z.*

5.3.4 Mapas de Karnaugh

Los mapas de Karnaugh constituyen una representación gráfica de las funciones de conmutación. Cualquier función de una variable se puede representar utilizando el mapa de Karnaugh correspondiente, que se da en la Figura 5.10a: consta de dos casillas, correspondientes a los dos posibles valores de la variable. Poniendo en cada casilla el valor 0 ó 1 que la correspondiente función asigne, se tiene un medio de representar las cuatro funciones de una variable. En la Figura 5.10b, c y d se tiene los mapas de Karnaugh para dos, tres y cuatro variables, respectivamente. Cada casilla corresponde a un minterm; en el ángulo inferior izquierdo de cada casilla se ha escrito el valor decimal del minterm correspondiente.

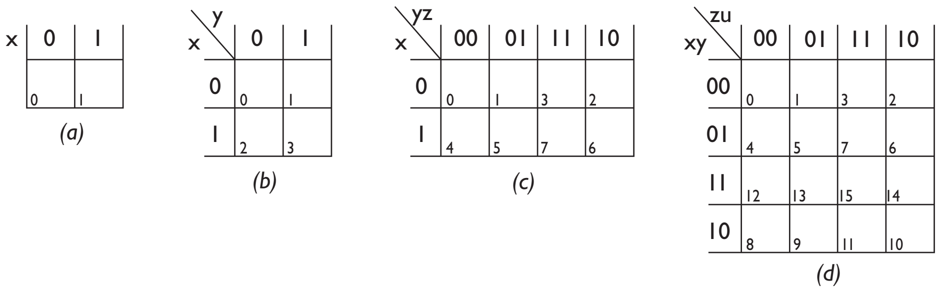


Figura 5.10. *Mapas de Karnaugh para: (a) una variable; (b) dos variables; (c) tres variables; (d) cuatro variables.*

En el caso de tres variables, las columnas se han asignado de forma que cada columna sea adyacente a sus dos vecinas, entendiendo que la primera y la última son vecinas. Por ejemplo, la columna 11 es adyacente a las 01 y 10; la 10 es adyacente a 11 y 00; y así sucesivamente. Esto mismo se ha hecho para las filas y columnas del mapa de cuatro variables. De forma similar se construyen mapas de Karnaugh para más variables. En la Figura 5.11 se tienen las representaciones en forma de mapa de Karnaugh de las funciones f_1 y f_2 de la Figura 5.7a y de la función F de la Figura 5.9a (únicamente se ha especificado los valores 1; las casillas en blanco corresponden a los ceros).

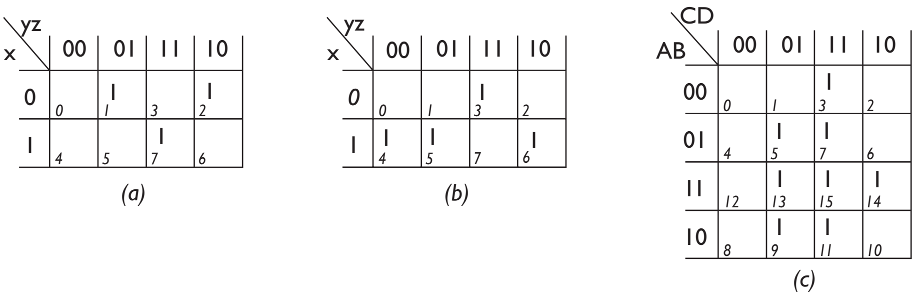


Figura 5.11. *Mapas de Karnaugh para (a) f_1; (b) f_2, y (c) F.*

El principal interés de los mapas de Karnaugh reside en su posible utilización en la minimización de funciones booleanas. Cada celda de un mapa de Karnaugh de n variables puede caracterizarse con n coordenadas. Por ejemplo, la celda 5 del mapa de Karnaugh de tres variables tiene como coordenadas 101 ($x = 1$, $y = 0$, $z = 1$) y corresponde al minterm $x\overline{y}z$. Dos

celdas (o cubos de orden cero o adyacencias de orden cero) de un mapa de Karnaugh son adyacentes y forman un cubo de orden uno si las coordenadas de ambas celdas son idénticas salvo una, que en un cubo valdrá cero y en otro valdrá uno. Un cubo de orden uno de n variables puede darse con $(n - 1)$ coordenadas binarias y una coordenada vacua, para la que se utilizará el símbolo −. Por ejemplo, los cubos de orden cero para cuatro variables 0011 y 0001 son adyacentes, y forman un cubo de orden uno de coordenadas 00 − 1. Un cubo de orden uno (o adyacencia de primer orden) corresponde a un producto de $(n - 1)$ variables, faltando la variable correspondiente a la coordenada vacua. Por ejemplo, al cubo 00 − 1 le corresponde el producto *xyu*, como fácilmente se comprueba:

$$\overline{x}\,\overline{y}\,\overline{z}u + \overline{x}\,\overline{y}zu = \overline{x}\,\overline{y}\,(\overline{z} + z)u = \overline{x}\,\overline{y}u$$

En general, dos cubos de orden i son adyacentes y forman un cubo de orden $i + 1$ si ambos tienen coordenadas vacuas en las mismas posiciones, y las coordenadas no vacuas son todas iguales salvo una; el cubo de orden $i + 1$ resultante tendrá $i + 1$ coordenadas vacuas: las i de los cubos adyacentes y otra en la posición en la que difieren las coordenadas de los cubos adyacentes. En este caso se dice que el cubo de mayor orden cubre a aquellos de los que procede.

Dada una función de n variables, se dice que una adyacencia de cualquier orden de esas variables es un implicante de la función si para aquellos valores de las variables para los que la adyacencia toma el valor uno, también la función toma el valor uno. Por ejemplo, la adyacencia *BCD* es un implicante de la función *F* del Ejemplo 5.9.

EJEMPLO 5.11

Simplificar la función F del Ejemplo 5.8.

Solución: *en la Figura 5.11c se representa F en un mapa de Karnaugh, que se repite en la Figura 5.12a. En esta misma figura se indican las agrupaciones de celdas correspondientes a la aplicación del algoritmo. En la Figura 5.12b se dan las coordenadas de cada cubo de orden cero que forman cada cubo de mayor orden, así como las adyacencias correspondientes.*

Cada 1 de F se puede utilizar múltiples veces para construir adyacencias de mayor orden. Por ejemplo, la celda 1111 se utiliza en cuatro ocasiones. Para sintetizar F hay que cubrir todos sus unos. Teniendo en cuenta esto último, es inmediato obtener una cobertura mínima para F como suma de implicantes primos; basta fijarnos en que cada una de las cuatro adyacencias, a, b, c, y d, incluye un 1 no cubierto por ninguna otra adyacencia, con lo que es imprescindible incluir las cuatro adyacencias en la realización de F, y con las cuatro queda cubierta F; es decir:

$$F = a + b + c + d = CD + BD + AD + ABC$$

Como era de esperar, resulta para F la misma expresión obtenida en el Ejemplo 5.9.

EJEMPLO 5.12

Utilizando el mapa de Karnaugh, obtener las adyacencias de las funciones ***u***, ***v*** *y* ***w*** *del Ejemplo 5.10.*

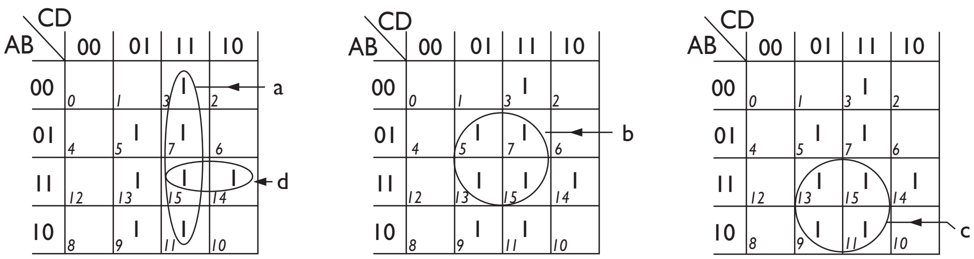
Solución: *en la Figura 5.13 se tienen los mapas de Karnaugh de las tres funciones, marcando las adyacencias de cada una.*

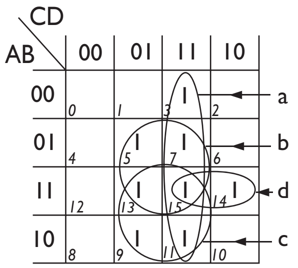
Se dice que un implicante de una función es un **implicante primo** si las adyacencias que resultan del implicante eliminando una cualquiera de las variables ya no son implicantes de la función. Por ejemplo, la adyacencia *BCD* (Ejemplo 4.9) no es un implicante primo de *F*, dado que la adyacencia *BD* o la *CD* son implicantes de *F*.

Una vez introducidos todos estos conceptos, es inmediato presentar un algoritmo para simplificar funciones de conmutación utilizando el mapa de Karnaugh. Una vez representada la función a simplificar en un mapa de Karnaugh, el algoritmo consta de los pasos siguientes:

a) Escribir todos los cubos (y las adyacencias correspondientes) de cualquier orden tales que cada uno cumpla las dos condiciones siguientes:
   a1) todas las celdas del cubo corresponden a valores 1 de la función.
   a2) no está cubierto por ningún otro cubo que cumpla a1).
b) Con las adyacencias obtenidas en a), construir una cobertura mínima para todos los unos de la función. La forma de obtener esa cobertura mínima va a ser intuitiva, aunque es fácil sistematizarla.

Las adyacencias obtenidas en a) son los implicantes primos de la función de que se trate, como se puede comprobar en cada caso.





(a)

a: 0011, 0111, 1111, 1011 → – – 11: *CD*
b: 0101, 0111, 1101, 1111 → – 1 – 1: *BD*
c: 1101, 1111, 1001, 1011 → 1 – – 1: *AD*
d: 1111, 1110 → 1 1 1 –: *ABC*

(b)

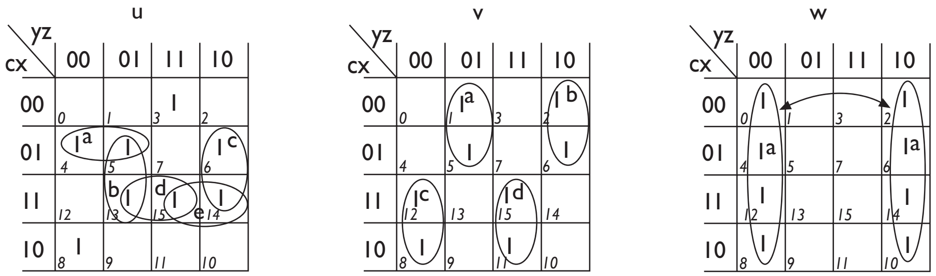
Figura 5.12. *(a) Mapa de Karnaugh de F, indicando las agrupaciones de celdas; (b) adyacencias que resultan.*

EJEMPLO 5.13

Simplificar la función f dada en el mapa de Karnaugh de la Figura 5.14a.

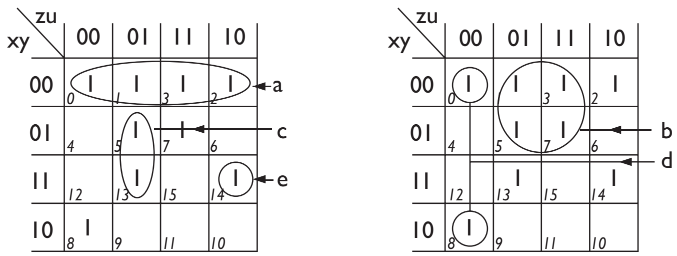
Solución: *en la Figura 5.14b se dan las adyacencias que resultan al agrupar adecuadamente los cubos con valor 1. Es inmediato comprobar que las cinco adyacencias deben aparecer en la realización de f; es decir*

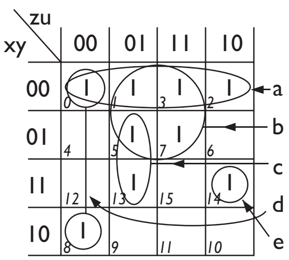
$$f = a + b + c + d + e = \overline{x}\,\overline{y} + \overline{x}u + y\overline{z}u + \overline{y}\,\overline{z}\,\overline{u} + xyz\overline{u}$$



u	v	w
a: 4 – 5 → $\bar{c}x\bar{y}$	a: 1 – 5 → $\bar{c}\,\bar{y}\,z$	a: 0 – 2 – 4 – 6 – 8 – 10 – 12 – 14: $\bar{z}$
b: 5 – 13 → $x\bar{y}\,z$	b: 2 – 6 → $\bar{c}y\bar{z}$	
c: 6 – 14 → $xy\bar{z}$	c: 8 – 12 → $c\bar{y}\,\bar{z}$	
d: 13 – 15 → cxz	d: 11 – 15 → cyz	
e: 14 – 15 → cxy		

Figura 5.13. *Adyacencias de las funciones del Ejemplo 5.12.*





(a)

a:	0000, 0001, 0011, 0010	→	00 – –: $\bar{x}\,\bar{y}$
b:	0001, 0011, 0101, 0111	→	0 – – 1: $\bar{x}\,u$
c:	0101, 1101	→	– 1 0 1: $y\,\bar{z}\,u$
d:	0000, 1000	→	– 0 0 0: $\bar{y}\,\bar{z}\,\bar{u}$
e:	1110	→	: $x\,y\,z\,\bar{u}$

(b)

Figura 5.14. *(a) Mapa de Karnaugh y (b) adyacencias correspondientes al Ejemplo 5.13.*

EJEMPLO 5.14

Simplificar la función g dada en la Figura 5.15a.

Solución: *en este caso resultan seis adyacencias de orden uno. De ellas, necesariamente han de aparecer en la realización de g las adyacencias a, b y f, pues cubren algún 1 no cubierto por otra adyacencia. Con estas tres se cubren todas las celdas al valor 1, salvo la celda 1101; para cubrir esta celda puede utilizarse la adyacencia d o la e. Es decir*

$$g = a + b + f + d \qquad \text{o bien} \qquad g = a + b + f + e$$

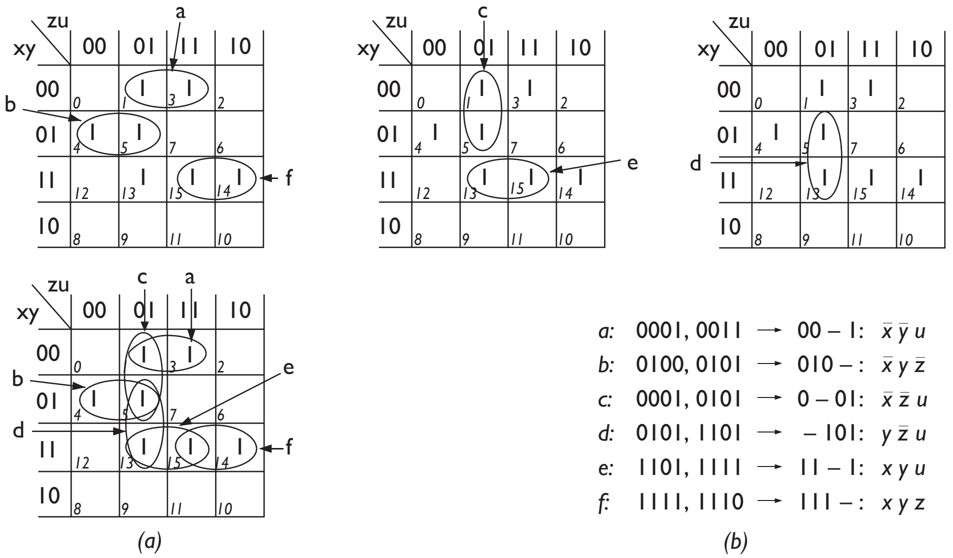


Figura 5.15. *Minimización de la función del Ejemplo 5.14: (a) Mapa de Karnaugh; (b) adyacencias resultantes.*

El mapa de Karnaugh se puede utilizar de forma similar para la realización en forma de producto de sumas, tal como se hace en el siguiente ejemplo.

EJEMPLO 5.15

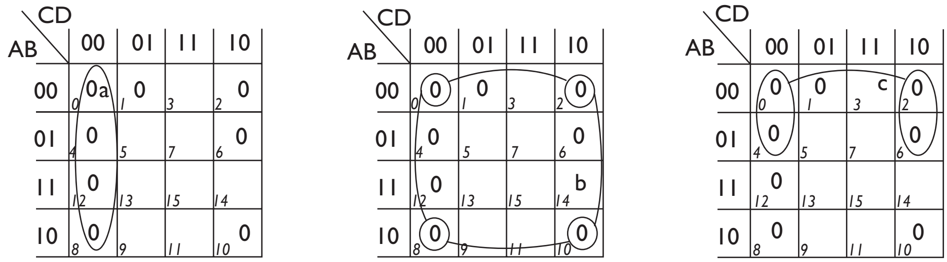
Simplificar, en la forma de producto de sumas, la función F del Ejemplo 5.8.

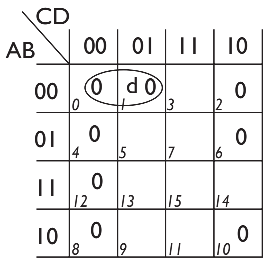
Solución: *en la Figura 5.11c se representa F en un mapa de Karnaugh, que se repite en la Figura 5.16a, pero dando los ceros en vez de los unos. En esta misma figura se indican las agrupaciones de celdas que pueden obtenerse tras la aplicación del algoritmo para el caso de sumas (no de productos). En la Figura 5.16b se dan las coordenadas de cada celda de orden cero que forman agrupaciones de mayor orden, así como las sumas correspondientes a esas agrupaciones.*

Cada 0 de F se puede utilizar múltiples veces para construir agrupaciones de mayor orden. Por ejemplo, la celda 0000 se utiliza en cuatro ocasiones. Para realizar F como producto de sumas hay que cubrir todos sus ceros. Teniendo en cuenta esto último, es inmediato obtener una cobertura mínima para F como producto de sumas; basta fijarnos en que cada una de las cuatro agrupaciones de la Figura 5.16b incluye un 0 no cubierto por ninguna otra, con lo que es imprescindible incluirlas todas en la realización de F, y con las cuatro queda cubierta F; es decir:

$$F = a \cdot b \cdot c \cdot d = (C + D) \cdot (B + D) \cdot (A + D) \cdot (A + B + C)$$

En otras palabras, F se puede sintetizar de forma mínima con una estructura O a Y utilizando cuatro puertas O (tres de dos entradas y una de una entrada) y una puerta Y de cuatro entradas. Comparando esta expresión para F con la obtenida en los Ejemplos 5.9 y 5.11, y teniendo en cuenta que las puertas Y y O con iguales entradas tienen el mismo coste, resulta que las dos realizaciones obtenidas para F son equivalentes. No siempre ocurre así: en ocasiones una de las dos formas de realización tiene un coste ligeramente superior a la otra. Pero por anticipado no se puede predecir qué realización es ventajosa; en cualquier caso la diferencia siempre es pequeña, normalmente como mucho se necesita una puerta más en una forma de realización que en la otra. Por ello se suele utilizar sólo una de ellas, concretamente la de suma de productos.





(a)

a: 0000, 0100, 1100, 1000 ⟶ *C + D*
b: 0000, 0010, 1000, 1010 ⟶ *B + D*
c: 0000, 0010, 0100, 0110 ⟶ *A + D*
d: 0000, 0001 ⟶ *A + B + C*

(b)

Figura 5.16. *(a) Mapa de Karnaugh de F, dando los ceros e indicando las agrupaciones de celdas; (b) sumas que resultan.*

5.4 Bloques funcionales combinacionales

En este apartado vamos a describir con cierto detalle algunos bloques funcionales combinacionales que están comercialmente disponibles en forma integrada y que se utilizan como bloques constructivos en el diseño de sistemas digitales.

5.4.1 Puertas combinacionales integradas

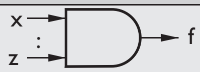
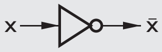
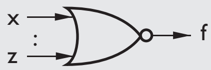
La tecnología de fabricación de circuitos lógicos permite integrar muchos circuitos en un mismo chip. Por esta razón, y dado que las puertas fundamentales se utilizan en gran número incluso en los sistemas digitales más sencillos, no se construyen por separado, sino que se incluyen varias puertas dentro de un mismo chip. Por ejemplo, en un solo chip puede tenerse alguna de las siguientes combinaciones de puertas Y (u O, o NO-Y, o NO-O):

- 1 puerta de 8 entradas
- 2 puertas de 4 entradas
- 3 puertas de 3 entradas
- 4 puertas de 2 entradas
- 5 inversores
- etcétera

Las puertas NO-Y (o NAND en inglés) y las NO-O (o NOR) son importantes como elementos de diseño, pues tanto la función NAND como la función NOR constituyen un conjunto suficiente de conectores. Esto quiere decir que para sintetizar cualquier función combinacional basta utilizar sólo puertas NAND (o sólo puertas NOR), en vez de utilizar tres tipos de puertas (Y, O y NO). Es claro el interés práctico de esta forma de abordar la

síntesis, pues es suficiente disponer de un solo tipo de puertas combinacionales. En la Tabla 5.2 se resume el comportamiento de todas las puertas lógicas mencionadas. Además de las puertas anteriores, también es de uso frecuente la puerta EXOR, cuya operación se simboliza con ⊕, y su complementaria, la puerta EXNOR, incluidas ambas en la Tabla 5.2.

Tabla 5.2. *Comportamiento de diversas puertas lógicas.*

Puerta	Símbolo	Función	Descripción
Y (AND)		$F = x \cdot \ldots \cdot z$	F = 1 si y sólo si todas las entradas a 1
O (OR)		$F = x + \ldots + z$	F = 0 si y sólo si todas las entradas a 0
NO (NOT)		$F = \overline{x}$	F es el complemento de la entrada
NO Y (NAND)		$F = \overline{x} + \ldots + \overline{z}$	F = 0 si y sólo si todas las entradas a 1
NO O (NOR)		$F = \overline{x} \cdot \ldots \cdot \overline{z}$	F = 1 si y sólo si todas las entradas a 0
O EXCLUSIVO (EXOR)		F = suma aritmética binaria de las entradas	F = 1 si y sólo si es impar el número de entradas a 1
NO O EXCLUSIVO (EXNOR)		F = complemento de la suma aritmética de las entradas	F = 1 si y sólo si es par el el número de entradas a 1

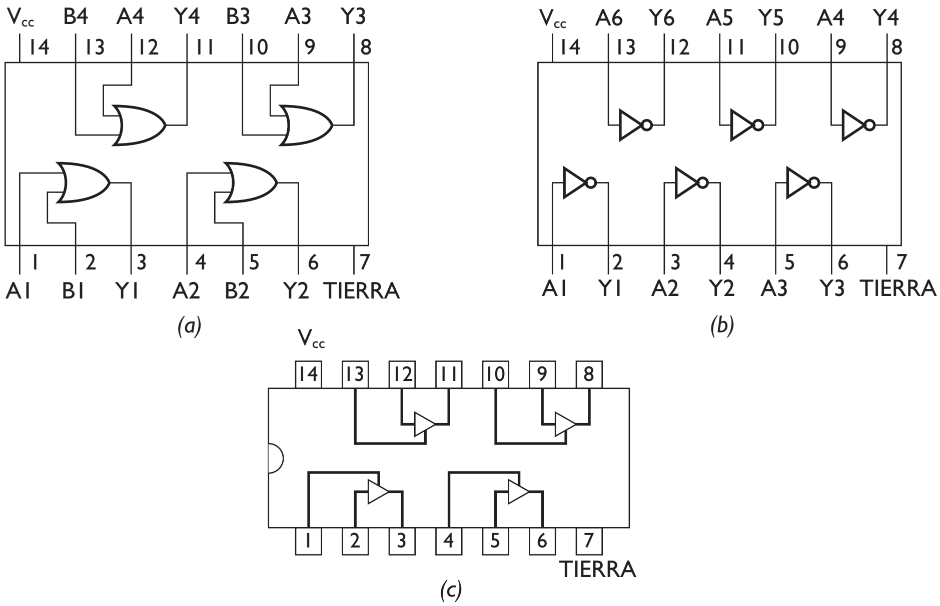


Figura 5.17. *Chips de puertas combinacionales: (a) 4 puertas O de dos entradas (7432); (b) 6 inversores (7404); (c) 4 adaptadores triestado (74126).*

En general la limitación en cuanto al número de puertas que se incluyan en un chip nace del número de patas necesarias para entradas, salidas, alimentación y tierra. En la Figura 5.17 se representan dos chips que incluyen diversas puertas combinacionales; el 7432 (Figura 5.17a) incluye cuatro puertas O de dos entradas cada una; el 7404 (Figura 5.17b) tiene seis inversores.

En los circuitos reales se utilizan en ocasiones elementos que no desempeñan papel alguno desde el punto de vista lógico, como es el caso del chip 74126 (Figura 5.17c), que incluye cuatro adaptadores. La finalidad de un adaptador es restaurar la señal de entrada a los valores físicos adecuados, pero coincidiendo la variable de salida con la de entrada, salvo el retardo temporal propio del dispositivo. Además, este adaptador es **triestado**; esto quiere decir que la salida puede estar en el *estado de baja impedancia,* pudiendo tomar los niveles de tensión correspondientes a los valores lógicos 0 ó 1, y puede estar también en *un estado de alta impedancia,* en el que todo ocurre como si ese punto estuviese físicamente desconectado del resto del circuito; el paso entre los estados de alta y de baja impedancia se controla con una entrada extra, y todo ocurre como si la puerta correspondiente tuviese tres estados (triestado): *0*, *1*, y *alta impedancia*. Hay otras puertas o bloques funcionales con salidas triestado. En todos los casos el paso entre los estados de alta y de baja impedancia se decide con una entrada de control. En el estado de baja impedancia la puerta o bloque funcional proporciona las salidas previstas; en el estado de alta impedancia es como si la salida estuviese desconectada del resto del circuito. Más adelante veremos su aplicación en la conducción de la información.

5.4.2 Sumador binario

Con frecuencia resulta necesario realizar operaciones aritméticas en los sistemas digitales; por ejemplo, es clara la necesidad en un computador de realizar sumas, restas, multiplicaciones y divisiones. Es conocido que, de estas cuatro operaciones, la única imprescindible es la suma, en el sentido de que, escogiendo adecuadamente la representación numérica y con los algoritmos correspondientes, las restantes operaciones se pueden realizar con sólo sumas. Así pues, a continuación nos vamos a concentrar en el diseño de un bloque combinacional que realice la suma binaria.

A continuación se va a diseñar un sumador para números binarios en los que se utiliza notación posicional. Supóngase que se tienen los números binarios de n bits $X = x_{n-1}\, x_{n-2} ... x_1\, x_0$ e $Y = y_{n-1}\, y_{n-2} ... y_1\, y_0$. Nuestro objetivo es obtener un sistema combinacional, al que llamaremos sumador binario, que nos permita sumar los bits x_i e y_i del mismo orden, de manera que asociando adecuadamente n sumadores binarios se pueda realizar la suma $X + Y$. Este bloque combinacional para sumar cada bit tendrá tres entradas (los bits x_i e y_i, mas el acarreo precedente, C_i) y dos salidas (la suma S_i y el acarreo para la etapa siguiente, C_{i+1}), tal como se representa en la Figura 5.18a. Es inmediato comprobar que la tabla verdad correspondiente a este sistema combinacional es la de la Figura 5.18b.

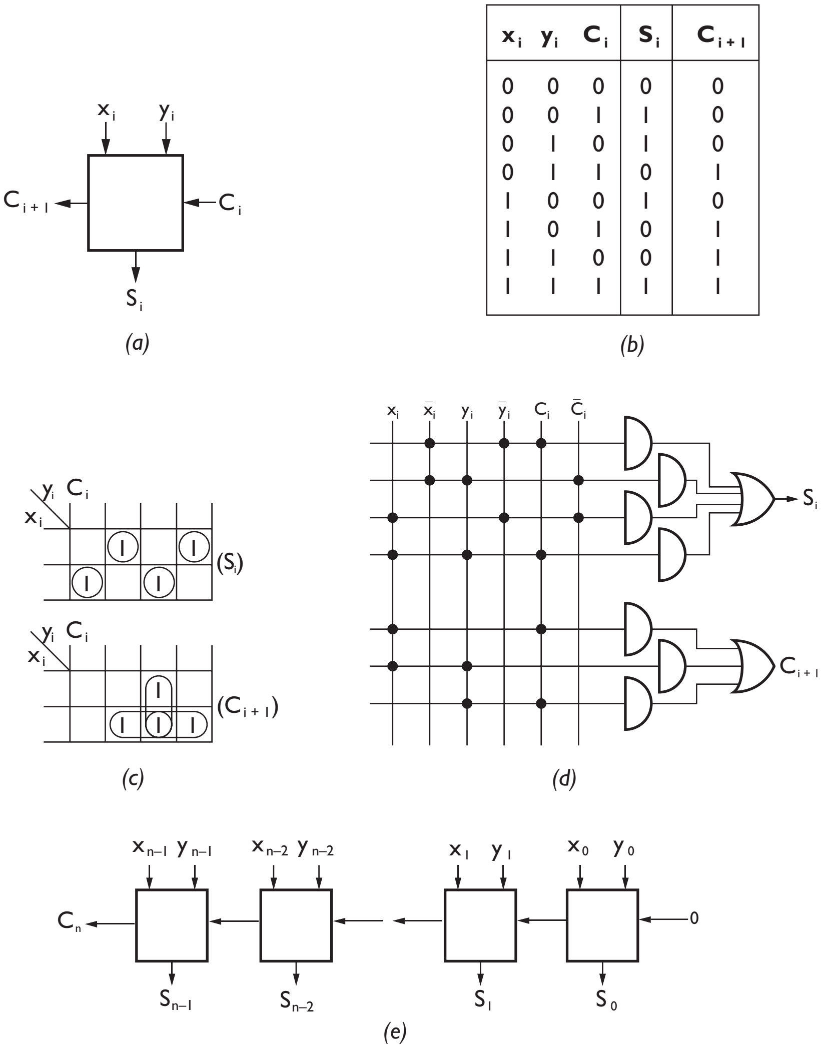
Representando S_i y C_{i+1} en forma de mapa de Karnaugh (Figura 5.18c) se concluye que las expresiones de ambas en forma de suma de productos es

$$S_i = \overline{x}_i \overline{y}_i C_i + \overline{x}_i y_i \overline{C}_i + x_i \overline{y}_i \overline{C}_i + x_i y_i C_i$$
$$C_{i+1} = x_i C_i + x_i y_i + y_i C_i$$

La suma S_i está dada utilizando puertas Y y O; si se utilizan puertas EXOR, entonces S_i es simplemente la función EXOR de las tres entradas; es decir, $S_i = x_i \oplus y_i \oplus C_i$.

El circuito correspondiente, con puertas Y y O, es el de la Figura 5.18d, en la que, para simplificar el dibujo, las diferentes entradas a cada puerta Y se han representado mediante una sola línea con múltiples conexiones (una por cada entrada).

Para sumar las palabras *X* e *Y* de *n* bits basta conectar *n* sumadores binarios en la forma representada en la Figura 5.18e; en este caso los acarreos se propagan en cascada a través de los *n* sumadores.



x_i	y_i	C_i	S_i	C_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Figura 5.18. *Sumador binario: (a) representación en forma de bloque; (b) tabla verdad; (c) mapas de Karnaugh para S_i y C_{i+1}; (d) realización Y a O; (e) red para sumar palabras de n bits.*

Hay disponibles en el mercado diversos chips que son sumadores binarios, aparte de la ALU que se verá posteriormente. Por ejemplo, el chip 74283 es un sumador binario para palabras de 4 bits, que admite un bit de acarreo de entrada y que, además de los 4 bits de la suma, genera un bit de acarreo de salida. Utilizando estos bits de acarreo es inmediato,

utilizando este chip 74283, construir sumadores de cualquier longitud (es decir, para sumandos de cualquier número de bits).

También hay disponibles chips para realizar otras operaciones aritméticas, como la multiplicación.

5.4.3 Comparadores

Una operación frecuente en el tratamiento de la información es la comparación de dos datos. Por ejemplo, al ordenar de mayor a menor una tabla de números o al ordenar alfabéticamente una serie de palabras, se van comparando dos a dos y se ordenan en consecuencia. Para realizar esta operación se utilizan los comparadores, cuya operación se va a describir basándonos en uno concreto disponible como chip. Se trata del 7485, que tiene 16 patas y es un comparador de magnitud para caracteres de 4 bits. Tiene 8 entradas de datos agrupadas como $A = a_3a_2a_1a_0$ y $B = b_3b_2b_1b_0$ (los dos caracteres de 4 bits a comparar), y tres salidas, A>B, A = B y A<B, correspondientes a los tres posibles resultados. Además tiene tres entradas ($A>B_e$, $A = B_e$ y $A<B_e$) para facilitar la conexión en cascada de chips de este tipo, con lo que resulta inmediato construir comparadores para palabras de cualquier longitud, como se puede ver en la Figura 5.19.

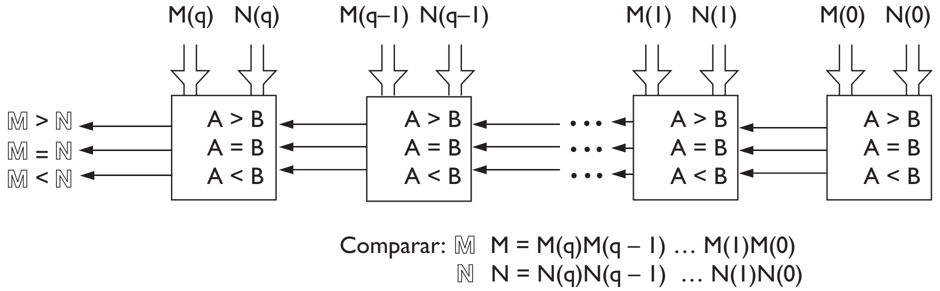


Figura 5.19. *Conexión en cascada de comparadores.*

5.4.4 Codificadores

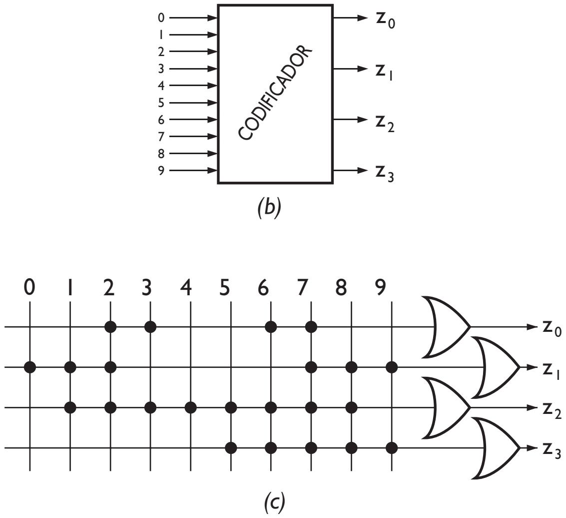
Un codificador es un dispositivo combinacional con *n* entradas y *m* salidas, tal que en un instante cualquiera sólo una de las entradas puede tomar el valor 1 (las restantes tomarán el valor 0). El codificador genera como salida una combinación de *m* bits que es única para cada entrada al valor 1, conociéndose esta combinación como código de ese carácter de entrada. Por ejemplo, en el código Gray los diez caracteres decimales se codifican tal como se da en la Figura 5.20a. En consecuencia, un codificador decimal en el código Gray tendrá diez entradas (de 0 a 9) y cuatro salidas (Figura 5.20b); de las diez entradas, sólo una podrá tomar el valor 1 en un instante cualquiera, y a la salida se tendrá el código $z_3z_2z_1z_0$ correspondiente a esa entrada que esté al valor 1. Es inmediato que para construir un codificador basta utilizar *m* puertas O, tal como se representa en la Figura 5.20c para el codificador Gray que se está sintetizando como ejemplo, supuesto que de alguna forma se garantiza que sólo una de las entradas está al valor 1. En ocasiones un codificador se representa tal como en la Figura 5.20d.

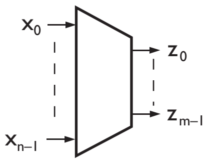
Los codificadores más usuales son los binarios, que tienen 2^n entradas y *n* salidas. En cada instante generan como salida el código binario correspondiente a la entrada activa. Por

ejemplo, un codificador binario con 4 entradas y 2 salidas realiza la codificación dada en la Figura 5.20e y se simboliza tal como se representa en la Figura 5.20f.

Carácter	z_3	z_2	z_1	z_0
0	0	0	1	0
1	0	1	1	0
2	0	1	1	1
3	0	1	0	1
4	0	1	0	0
5	1	1	0	0
6	1	1	0	1
7	1	1	1	1
8	1	1	1	0
9	1	0	1	0

(a)

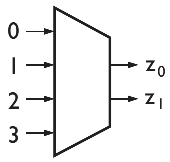




(d)

	z_1	z_0
0	0	0
1	0	1
2	1	0
3	1	1

(e)



(f)

Figura 5.20. *(a) Tabla del código decimal Gray; (b) símbolo para el codificador decimal Gray; (c) realización del código Gray; (d) símbolo del codificador; (e) tabla del codificador binario con dos salidas; (f) símbolo del codificador binario con dos salidas.*

Por extensión, usualmente también se denominan codificadores los dispositivos que efectúan una traslación de códigos. Por ejemplo, pueden trasladar del código Gray al binario.

Hay disponibles en forma integrada varios codificadores. Un ejemplo es el chip 74148 (de 16 patas), que es un codificador binario de 8 a 3 (8 entradas y 3 salidas); otro ejemplo es el chip 74147 (de 16 patas), que es un codificador BCD, de 10 a 4.

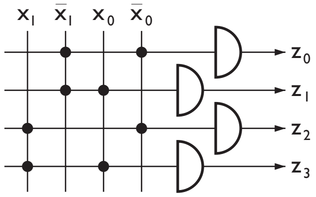
5.4.5 Decodificadores/demultiplexores

Un decodificador es un dispositivo combinacional con n entradas y m salidas y funciona de manera que cada combinación de valores en las entradas pone a 1 una y sólo una de las salidas, permaneciendo las restantes salidas al valor cero (0). Es decir, un decodificador, como su propio nombre indica, realiza la operación inversa de un codificador. En realidad lo importante en la actuación de un decodificador es que diferencia una y sólo una de sus salidas de entre todas las demás, poniéndola al valor 1 mientras las restantes permanecen al valor 0, o también poniéndola al valor 0 mientras las otras salidas están al valor 1. Por tanto

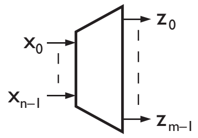
se puede diseñar un decodificador decimal en el código Gray, que realizará la operación inversa del codificador Gray visto anteriormente; este decodificador tendrá cuatro entradas y diez salidas.

x_1	x_0	z_3	z_2	z_1	z_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

(a)



(b)



(c)

Figura 5.21. *(a) Tabla verdad del decodificador binario de 1 entre 4; (b) realización de este decodificador; (c) símbolo de un decodificador.*

Dado que todos los decodificadores se diseñan utilizando el mismo procedimiento, vamos a considerar únicamente los decodificadores binarios. Un **decodificador binario** tiene n entradas y 2^n salidas, que podemos considerar numeradas desde 0 hasta $2^n - 1$, de manera que cada combinación de entradas excita (pone a 1 si las demás están a 0, o pone a 0 si las demás están al valor 1) la salida correspondiente a su valor en binario. El decodificador binario de 1 entre 4 tiene dos entradas y cuatro salidas; su funcionamiento queda descrito mediante la tabla verdad de la Figura 5.21a. Como se ve en esta tabla verdad, las cuatro funciones de salida son los cuatro minterms de dos variables, por lo que este decodificador se puede construir con cuatro puertas Y (una por cada salida), tal como se representa en la Figura 5.21b. El símbolo que en ocasiones se utiliza para un decodificador es el de la Figura 5.21c. Cualquier decodificador binario con n entradas y 2^n salidas se puede construir con 2^n puertas Y.

Hay disponibles en forma integrada múltiples chips decodificadores. Por ejemplo, el chip 74138 (16 patas), es un decodificador binario de 3 a 8; el chip 74154 (24 patas) es otro decodificador binario de 4 a 16; el chip 7442 (16 patas) es un decodificador decimal, de 4 a 10.

Un **demultiplexor** es un dispositivo combinacional con una entrada de datos, x, con n entradas de control, $y_0,...,y_{n-1}$, y con 2^n salidas de datos, $z_0,...,z_{2^n-1}$. Funciona de manera que en un instante cualquiera la entrada x se conecta a una y sólo una de las salidas, determinada dicha salida por la combinación de las señales de control: concretamente se selecciona la salida z_i tal que las entradas de control tienen el valor i en binario. Los demultiplexores, según esta descripción, son dispositivos para la conducción de la información; es decir, la entrada x no sufre modificación alguna y con el demultiplexor únicamente se controla el camino que en cada caso ha de seguir la señal aplicada a la entrada. Por ejemplo, un demultiplexor de *1 a 4* tiene una entrada de datos, x, dos entradas de control, y_1 e y_0, y cuatro salidas de datos, z_3, z_2, z_1, z_0; su funcionamiento queda descrito mediante la tabla de la Figura 5.22a. El símbolo que se suele utilizar para un demultiplexor es el de la Figura 5.22b.

Es inmediato que si se le añade una nueva entrada x al decodificador de la Figura 5.21b, de manera que esta entrada adicional lo sea de todas las puertas Y, y las entradas x_1 y x_0 pasan a ser las entradas de control y_1 e y_0, tal como se representa en la Figura 5.22c, el decodificador se convierte en un demultiplexor; es decir, el mismo circuito de la Figura 5.22c puede actuar como un demultiplexor de *1 a 4*, o como un decodificador con cuatro salidas (para esto último basta hacer $x = 1$).

Como ejemplo de chip decodificador/demultiplexor se puede mencionar la 74155 (de 16 patas), que incluye dos decodificadores/demultiplexores de *2 a 4* líneas, compartiendo los dos decodificadores/demultiplexores las señales de selección.

y_1	y_0	z_3	z_2	z_1	z_0
0	0	0	0	0	x
0	1	0	0	x	0
1	0	0	x	0	0
1	1	x	0	0	0

(a)

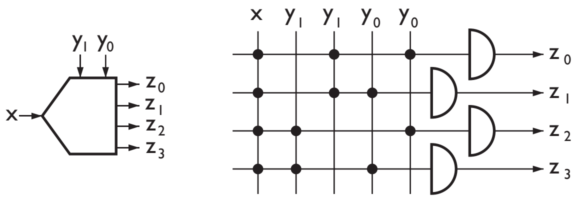


Figura 5.22. *Demultiplexor de 1 a 4: (a) tabla verdad; (b) símbolo; (c) realización.*

Utilizando demultiplexores con n entradas de control se puede construir muy fácilmente cualquier demultiplexor con $p \cdot n$ entradas de control (p entero). Para ello basta construir una estructura en árbol tal como la de la Figura 5.23a con p niveles, en la que todos los demultiplexores del mismo nivel estén controlados por las mismas señales de control. Por ejemplo, si se dispone de demultiplexores de *1 a 2*, y se desea construir un demultiplexor de *1 a 8*, se puede hacer con la red de la Figura 5.23a, que consta de tres niveles con 1, 2 y 4 demultiplexores, respectivamente. El demultiplexor del primer nivel está controlado por y_2; los demultiplexores del segundo nivel están controlados ambos por y_1; los demultiplexores de tercer nivel están controlados todos ellos por y_0. Desde el punto de vista de las señales externas, estos siete demultiplexores actúan como si se tratase de un único demultiplexor de *1 a 8*.

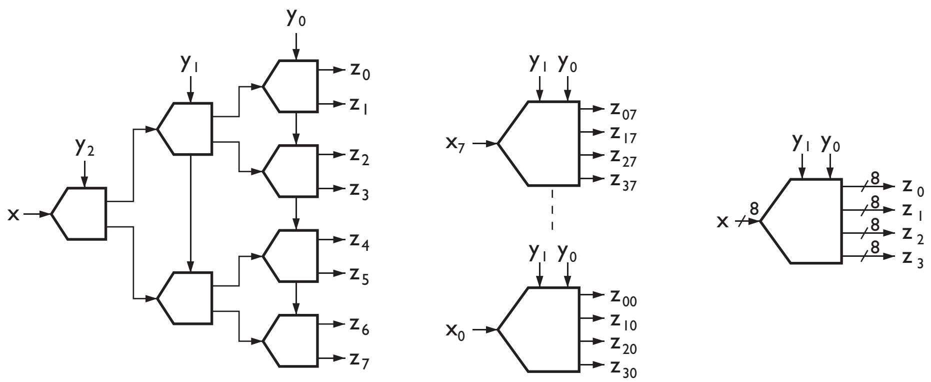


Figura 5.23. *(a) Asociación de demultiplexores de 1 a 2 para construir un demultiplexor de 1 a 8; (b) asociación de demultiplexores de bit para construir un demultiplexor de byte; (c) representación de un demultiplexor de byte.*

Los demultiplexores mencionados hasta ahora son demultiplexores de bit; con frecuencia es necesario utilizar demultiplexores de palabras de r bits. Es inmediato que para construir un demultiplexor de *1 a n* para palabras de r bits basta utilizar r demultiplexores de bit de *1 a n*, de manera que todos estén controlados por las mismas señales de control. Por ejemplo, supóngase

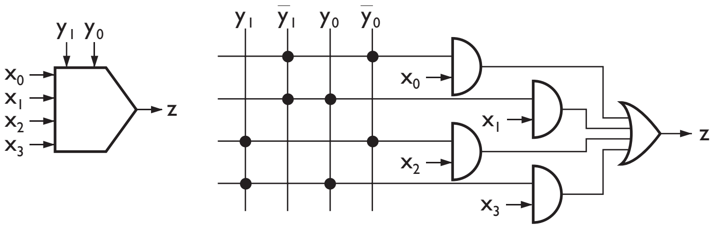
que se desea construir un demultiplexor de *1 a 4* para bytes, utilizando demultiplexores de bit de *1 a 4*. Si $X = x_7...x_0$ es el byte de entrada, $Z_0 = z_{07}...z_{00}...$, $Z_3 = z_{37}...z_{30}$ son los bytes de salida, e $y_1 y_0$ son las entradas de control, es inmediato que la red con 8 demultiplexores de la Figura 5.23b actúa de la forma deseada; este demultiplexor de palabra se representará como en la Figura 5.23c.

5.4.6 Multiplexores

Un multiplexor es un sistema combinacional para la conducción de la información, con n entradas de control, y_{n-1}, ..., y_0, 2^n entradas de datos, x_{2^n-1}, ..., x_0, y una salida z. Funciona de manera que en un instante cualquiera una y sólo una de las entradas de datos se conecta a la salida, determinándose mediante las entradas de control qué entrada de datos es la que en cada instante se conecta a la salida. Es decir, un multiplexor realiza la operación inversa a la de un demultiplexor. Por ejemplo, un multiplexor de *4 a 1* tiene cuatro entradas de datos, x_3, x_2, x_1, x_0, tiene dos entradas de control, y_1 e y_0, y una salida, z; su funcionamiento queda descrito mediante la tabla de la Figura 5.24a. El símbolo que a veces se utiliza para un multiplexor es el de la Figura 5.24b. Es fácil comprobar que el circuito de la Figura 5.24c es un multiplexor de *4 a 1*.

y_1	y_0	z_0
0	0	x_0
0	1	x_1
1	0	x_2
1	1	x_3

(a)



(b) *(c)*

Figura 5.24. *Multiplexor de 4 a 1; (a) tabla de funcionamiento; (b) bloque simbólico; (c) realización Y a O.*

Hay múltiples chips que incluyen multiplexores, como es el caso del 74157 (16 patas), que incluye 4 multiplexores de *2 a 1*, todos con la misma señal de selección, o el del 74251 (16 patas), que es un multiplexor de *8 a 1* con salida triestado.

Si se dispone de multiplexores con n entradas de control, se puede construir un multiplexor con $p \cdot n$ entradas de control (p entero). Para ello basta construir una estructura en árbol tal como la de la Figura 5.25a con p niveles, con todos los multiplexores de un mismo nivel controlados por las mismas señales de control, de forma idéntica a como se ha hecho en el caso de los demultiplexores. Esta técnica de ampliación de multiplexores se puede utilizar también de forma que en la red se incluyan multiplexores con números diferentes de señales de control (esto se puede trasladar inmediatamente a las asociaciones de demultiplexores). Por ejemplo, supóngase que se dispone de multiplexores de *2 a 1* y de *4 a 1*, y se desea construir un multiplexor de *8 a 1*. Una primera solución consiste en construir una red con tres niveles, utilizando únicamente multiplexores de *2 a 1*, similar a la de la Figura 5.23a. Otra solución es utilizar una red con dos niveles, uno primero con dos multiplexores de *4 a 1*, y uno segundo con un multiplexor de *2 a 1*, tal como se representa en la Figura 5.25a. También se puede utilizar una red con dos niveles, uno primero con cuatro multiplexores de *2 a 1*, y un segundo con un multiplexor de *4 a 1*, tal como se representa en la Figura 5.25b.

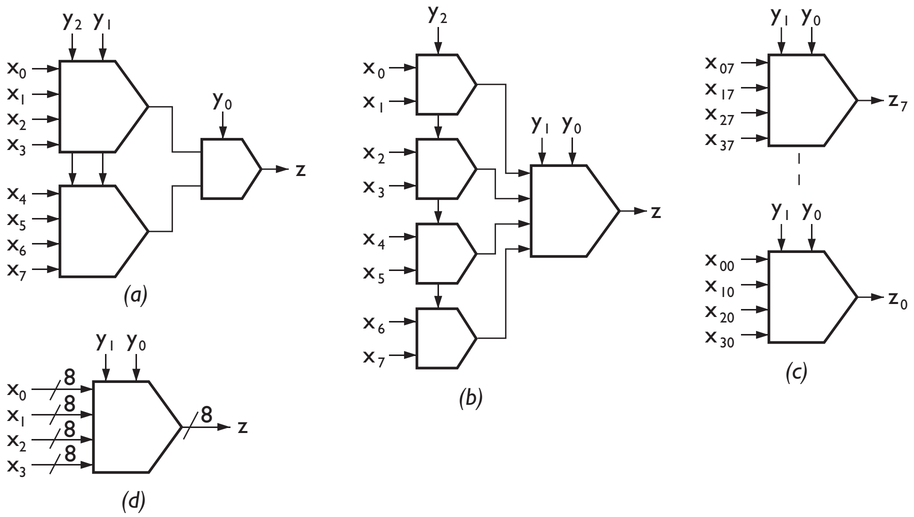


Figura 5.25. *Asociaciones de multiplexores: (a) de dos multiplexores de 4 a 1, y uno de 2 a 1 para formar otro de 8 a 1; (b) de cuatro multiplexores de 2 a 1, y uno de 4 a 1, para construir uno de 8 a 1; (c) de ocho multiplexores de 4 a 1 para formar un multiplexor de bytes de 4 a 1; (d) representación de un multiplexor de bytes de 4 a 1.*

Los multiplexores descritos son de bit y, lo mismo que se ha dicho para los demultiplexores, es inmediato construir un multiplexor de *r a 1* para palabras de *p* bits utilizando *p* multiplexores de bit de *r a 1* controlados por las mismas señales de control. En la Figura 5.25c se indica cómo construir un multiplexor de *4 a 1* para bytes con ocho multiplexores de *4 a 1* para bits; en la Figura 5.25d se da el símbolo utilizado para este multiplexor.

Además de servir para la conducción de la información, los multiplexores también pueden utilizarse para sintetizar funciones booleanas. Es inmediato que con un multiplexor con *n* entradas de control se puede sintetizar cualquier función booleana de *n* variables; para ello basta considerar las variables de control del multiplexor como variables de la función a sintetizar, y aplicar a las entradas de datos los valores constantes 0 ó 1 que la función a sintetizar aplique a cada combinación de sus variables. Por ejemplo, como un multiplexor de *4 a 1* tiene dos variables de control, se puede sintetizar cualquier función booleana de dos variables. Concretamente supóngase que se desea sintetizar la función $F(x,y)$ dada en forma de tabla verdad en la Figura 5.26a. Es inmediato comprobar que con el multiplexor de la Figura 5.26b, cuyas entradas de control son x e y, y cuyas entradas de datos permanecen constantemente a los valores 0 ó 1, según se indica, se sintetiza dicha función.

También es fácil comprobar que un multiplexor con *n* entradas de control sirve para sintetizar cualquier función booleana de $n + 1$ variables. En este caso la función a sintetizar se descompone en 2^n subfunciones con respecto a *n* de sus variables. Estas *n* variables con respecto a las cuales se ha efectuado la descomposición se aplican a las entradas de control, y a las entradas de datos se aplican las subfunciones que corresponda. Es claro que estas subfunciones, al depender de una sola variable, *z* por ejemplo, pueden tomar uno de los cuatro valores siguientes: 0, 1, z, $\overline{z}$. Por ejemplo, supóngase que se desea sintetizar la función $F(x,y,z)$ dada en forma de tabla verdad en la Figura 4.26c, utilizando un multiplexor

de *4 a 1*. Descomponiendo esta función con respecto a las variables x e y resultan las siguientes subfunciones:

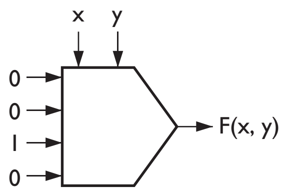
$$F(0,0,z) = 0; \qquad F(0,1,z) = 1; \qquad F(1,0,z) = \overline{z}; \qquad F(1,1,z) = z$$

Por consiguiente, a las dos entradas de control del multiplexor se le aplican las variables x e y; a las cuatro entradas de datos se le aplican estas cuatro subfunciones, según se representa en la Figura 5.26d.

Atendiendo a esta posibilidad de los multiplexores de sintetizar cualquier función de hasta un número determinado de variables, se dice que los multiplexores son **módulos lógicos universales**.

x	y	F
0	0	0
0	1	0
1	0	1
1	1	0

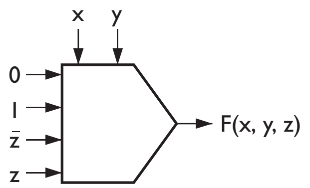
(a)



(b)

x	y	z	F	
0	0	0	0	0
0	0	1	0	
0	1	0	1	1
0	1	1	1	
1	0	0	1	$\overline{z}$
1	0	1	0	
1	1	0	0	z
1	1	1	1	

(c)



(d)

Figura 5.26. *Realización de funciones de conmutación con multiplexores: (a) función de dos variables a sintetizar; (b) realización de la misma con un multiplexor de 4 a 1; (c) función de tres variables a sintetizar; (d) realización de la misma con un multiplexor de 4 a 1.*

Si a las entradas de control de un multiplexor se aplican ordenadamente, a intervalos regulares, todos los diferentes 2^n caracteres de control, del 00...0 al 11...1, a la salida se tendrá, en instantes sucesivos, la información que se introduzca por todas las entradas, de la 0 a la 2^n–1. Utilizado de esta forma, es decir, para seleccionar secuencialmente datos, este dispositivo se conoce como **selector de datos**, denominación que a veces se utiliza en lugar de la de multiplexor.

Un multiplexor, utilizado como selector de datos, sirve para convertir información paralelo en información serie. Para ello basta con aplicar a las entradas correspondientes los 2^n bits del carácter paralelo que se desee transformar a serie, y a continuación, mediante las señales de control, se hace que a la salida aparezcan sucesivamente, en serie, los bits x_0, x_1, ..., x_{2^n-1} del carácter en cuestión. Utilizando esta misma idea, un demultiplexor podría servir para transformar información serie en información paralelo. Sin embargo, la transformación de serie a

paralelo se hace utilizando otros procedimientos, que se verán posteriormente, en el próximo capítulo (al estudiar los registros de desplazamiento), y no utilizando demultiplexores.

5.4.7 Acceso a buses

Una de las tareas a realizar en un sistema digital es la conducción de la información entre diferentes subsistemas. Para esta tarea se utilizan canales de conducción formados por *n* hilos, siendo *n* el número de bits que haya que transmitir simultáneamente; estos canales se conocen como **buses**.

Usualmente los subsistemas más relevantes de un sistema digital funcionan en paralelo, de manera que al decir que un sistema está organizado en palabras de un determinado número *n* de bits (ocho, por ejemplo), entre otras cosas se está indicando que la información desde un subsistema a otro se transmite por un **canal de información** o **bus** con esa misma anchura *n*.

La mayor velocidad en las comunicaciones entre los diferentes subsistemas se puede conseguir uniendo con un bus particular cada dos subsistemas entre los que deba haber comunicación. Ahora bien, si son muchos los subsistemas y entre ellos hay gran conectividad, se necesitarían muchos buses y la solución anterior resulta inviable. Lo que usualmente se hace es crear canales comunes de comunicación que conecten el mayor número posible de subsistemas, cada uno de ellos actuando como fuente o como destino de la información, y quizás en muchos casos el mismo subsistema actuando tanto de fuente como de destino, dependiendo del momento.

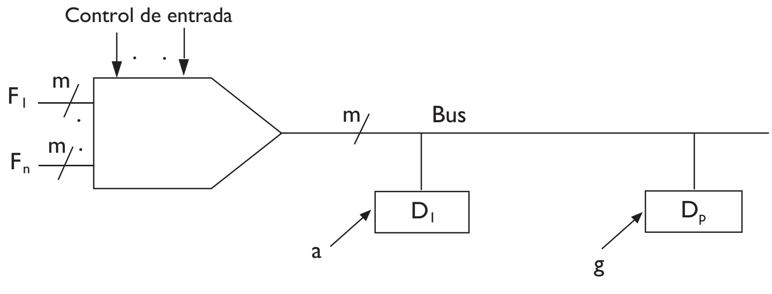


Figura 5.27. *Acceso a buses.*

En un determinado instante la información que se transmite por el bus provendrá de un subsistema concreto; es decir, la fuente de información en cada momento ha de ser necesariamente única. Esto implica que si dos subsistemas han de ser fuentes de información simultáneamente, dicha información ha de transmitirse por buses diferentes. También, si un bus tiene más de una fuente de información, la entrada al bus debe controlarse de forma que en cada instante se decida adecuadamente qué fuente se conecta al bus. Este control de entrada al bus se hace con un multiplexor de palabra, tal como se representa en la Figura 5.27; a las entradas de datos se aplican las diferentes fuentes de información y con las entradas de control del multiplexor de palabra se selecciona en cada instante qué fuente de información se aplica concretamente al bus. Esta misma función multiplexora se puede conseguir utilizando lógica triestado. Si las salidas de todas las fuentes de información son triestado (a cada una le corresponderá una señal de control para decidir el paso entre alta y baja impedancia), basta

garantizar, utilizando las señales de control de las salidas, que en un determinado instante sólo una de las salidas de las fuentes de información, como máximo, está en baja impedancia (es decir, conectada al bus).

En un determinado instante la fuente de información en un bus ha de ser única, pero obviamente el destino puede ser múltiple. Lo normal es que la información se escriba, en cada destino, en un registro individual, y lo que se hace para controlar la salida del bus es controlar individualmente el acceso a cada uno de los registros. Concretamente, tal como se representa en la Figura 5.27, cada destino de la información tendrá una señal de control con la que en cada instante se decidirá si capta o no la información que hay en el bus.

5.4.8 Memorias permanentes (ROM)

Una memoria organizada en 2^n palabras de m bits es una estructura lógica que tiene n entradas de dirección, m salidas de datos, y $m \cdot 2^n$ celdas de memoria; además puede tener m entradas de datos y p entradas de control. Esquemáticamente se representa tal como en la Figura 5.28a. Cada celda de memoria contiene un bit de información, que a través de las salidas de datos puede ser leído cuando se desee. Si la información contenida en cada celda se puede modificar (es decir, se puede "escribir" en la celda), la memoria correspondiente se denomina de lectura y escritura; en este caso, además de las n entradas de dirección, la memoria tiene m entradas de datos, a través de las cuales se introduce la información a escribir. Por el contrario, si la información contenida en cada celda no se puede modificar durante la operación normal de la memoria, es decir, es una información "permanente", la memoria se dice de sólo lectura. A continuación vamos a considerar únicamente las memorias de sólo lectura o memorias permanentes; las memorias de lectura y escritura se estudiarán más adelante.

A cada palabra de memoria le corresponde una "dirección" única, estando formada dicha dirección por una combinación específica de las n entradas de dirección. Para leer una palabra dada, en las entradas de dirección se aplica la dirección correspondiente a esa palabra, y en las salidas de datos aparece la información "escrita" en esa palabra de memoria.

Tras esta introducción el lector atento habrá concluido que cualquier red lógica combinacional con n entradas y m salidas se puede considerar como una memoria de sólo lectura con 2^n palabras de m bits, actuando las entradas de la red como entradas de dirección y las salidas de la red como salidas de datos. La información escrita en esa memoria es la correspondiente a las funciones de conmutación sintetizadas por la red combinacional. Por ejemplo, el multiplexor de la Figura 5.26b puede considerarse como una memoria de sólo lectura con cuatro palabras de 1 bit, siendo x e y las entradas de dirección y siendo (0, 0, 1, 0) la información escrita en esas cuatro palabras. De igual forma, el multiplexor de la Figura 5.26d puede considerarse como una memoria de sólo lectura con ocho palabras de 1 bit, siendo x,y,z las entradas de dirección, con la siguiente información escrita en las ocho palabras: 0, 0, 1, 1, 1, 0, 0, 1.

Aunque cualquier red lógica combinacional puede ser vista como una memoria permanente, esta denominación se suele reservar a las memorias **ROM** (Read Only Memory o **memorias de sólo lectura**). Una ROM es un sistema lógico combinacional con n entradas, $x_0,...,x_{n-1}$, y m salidas, $z_0,...,z_{m-1}$, que a cada carácter de entrada (de n bits) le hace corresponder de forma unívoca un carácter de salida (de m bits). Por tanto, una ROM puede considerarse como un dispositivo conversor de códigos y se puede imaginar como constituida por un decodificador seguido de un codificador, tal como se representa en la Figura 5.28b. Siguiendo

esta misma idea, en la Figura 5.28c se tiene una ROM formada por una matriz de puertas Y (decodificador) seguida de una matriz de puertas O (codificador), pudiéndose establecer a voluntad del diseñador las conexiones de las salidas de las puertas Y con las entradas de las puertas O. Estas $2^n \cdot m$ posibles conexiones son las celdas de memoria, de manera que "escribir" un 1 en una celda de memoria consiste en establecer una conexión entre la salida de una puerta Y y una entrada a una puerta O (las que correspondan a la celda en cuestión); para escribir un 0 (cero), no se establece la conexión. Una vez que se han fijado estos contactos, es decir, una vez construida la ROM, la información almacenada en la misma permanece indefinidamente, y se puede leer cuantas veces se desee.

La **escritura** de la ROM se hace de formas diferentes dependiendo de la tecnología de fabricación de los correspondientes chips. La puede hacer el fabricante de los CIs, en cuyo caso el usuario le indicará qué información quiere escribir, siendo ya inalterable una vez escrita. También hay chips que incluyen memorias ROM **programables** por el usuario, conocidos como **PROM** (P de programable). Incluso hay circuitos ROM que pueden borrarse una vez programadas, y volver a programarlos; son los chips **EPROM** (E de *erasable*: borrable). Los procesos de programación y borrado en las PROM o EPROM se realizan colocando los chips en dispositivos apropiados para estos fines, de manera que una PROM o una EPROM, una vez grabada y ya en su uso normal, no se diferencian en su funcionalidad de cualquier otra ROM. Una mayor versatilidad se consigue con las memorias **EEPROM** (PROM borrable eléctricamente), que no exigen el uso de dispositivos específicos para el borrado y la grabación sino que se pueden borrar y reescribir durante el uso de las mismas, aplicándoles sobretensiones adecuadas; es decir, el borrado y la grabación se hacen a una frecuencia mucho menor y aplicando unas tensiones diferentes a las del uso normal, pero sin tener que mover los chips de sus posiciones normales de funcionamiento. Un tipo particular de memorias EEPROM son las **memorias Flash**, en las que no se pueden borrar y regrabar posiciones individuales de las mismas, sino que se han de borrar y regrabar en bloques específicos, normalmente del tamaño de un chip.

Evidentemente, una ROM es una memoria permanente cuyos bits de dirección son las entradas de la ROM y cuyas salidas de datos son las m salidas de la ROM. Con frecuencia para las ROMs se utiliza la representación dada en la Figura 5.28d, que incluye 2^n filas de celdas, con m celdas cada una, escribiéndose en cada celda el valor 0 ó 1 grabado en las mismas.

Dada una ROM de un determinado tamaño; es decir, con un número de palabras de una longitud dada, es fácil construir otra ROM con palabras de mayor longitud o con mayor número de palabras, o ambas cosas. Esta **ampliación** del tamaño **de una ROM** se verá en la Sección 6.6 (en el que se estudian las memorias RAM), pues se hace igual que para una RAM.

Una ROM es un **módulo lógico universal** que permite sintetizar simultáneamente m funciones de conmutación de n variables. Para utilizarla de esta forma basta escribir la información por columnas en vez de por filas, de manera que en cada columna se escriben los valores 0 ó 1 de la tabla verdad de la correspondiente función a sintetizar. En la Figura 5.28e se tiene una ROM con la que se sintetizan las seis funciones $z_0, ..., z_5$ de las variables x_0, x_1, x_2. Es claro que con el decodificador se sintetizan los 2^n minterms de las n variables de entrada; con las conexiones entre las puertas Y y las puertas O se seleccionan los minterms que se incluyen en cada función de salida.

Además de su aplicación como módulos lógicos universales o como dispositivos conversores de códigos, las memorias ROM tienen múltiples aplicaciones para crear tablas de valores,

para generar secuencias específicas de caracteres o, en sistemas programables, para grabar rutinas de uso frecuente o bloques muy relevantes del sistema operativo.

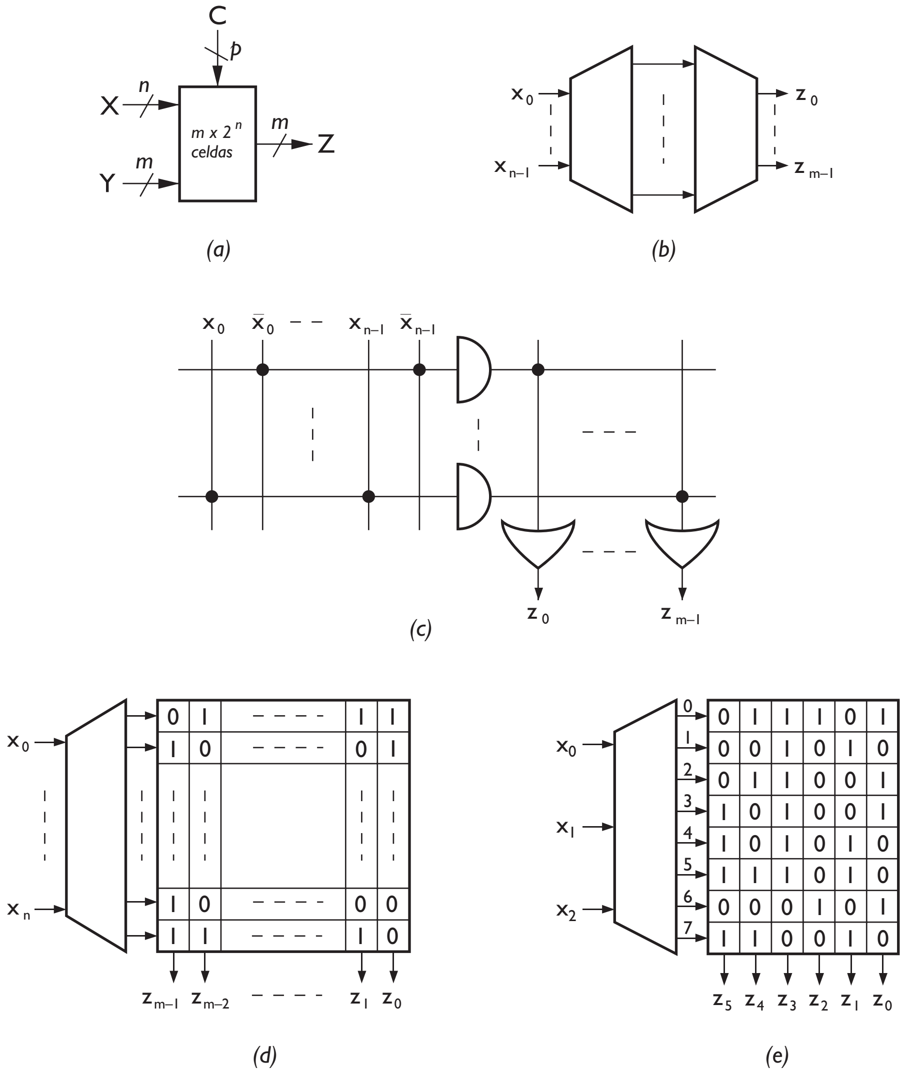


Figura 5.28. *(a) Representación esquemática de un bloque de memoria; (b) la ROM como asociación de un decodificador y un codificador; (c) realización Y a O de una ROM; (d) otra representación para las memorias ROM; (e) memoria ROM sintetizadora de seis funciones de tres variables.*

5.4.9 PLAs y PALs

Una ROM con n entradas y m salidas está formada por una matriz de puertas Y con 2^n puertas, de manera que a la salida de esta matriz de puertas Y se tienen sintetizados todos los minterms de las n entradas. Las conexiones entre las entradas y la matriz de puertas Y son fijas y corresponden a este objetivo de sintetizar todos los minterms, de manera que

si se añade una entrada (pasan de n a $n + 1$), se ha de duplicar el número de puertas Y (pasan de 2^n a $2^{n+1}=2 \cdot 2^n$). La matriz de puertas Y va seguida de una matriz con m puertas O, siendo programables las conexiones entre las salidas de las puertas Y (decodificador binario) y las entradas de las puertas O (codificador de salida), tal como se representa en la Figura 5.29a.

En muchas aplicaciones sería deseable tener la posibilidad de programación de las ROMs, pero sin la necesidad de sintetizar todos los minterms de las n entradas. Siguiendo esta idea se han diseñado estructuras programables similares a las ROMs: son las PLAs y las PALs. Ambas estructuras constan, al igual que las ROMs, de unas entradas externas que actúan sobre una matriz de puertas Y seguida de otra de puertas O, que producen las salidas; se diferencian en las posibilidades de programación.

En una **PAL** (***P****rogrammable* ***A****rray* ***L****ogic*) hay n entradas seguidas de una matriz de puertas Y (con un número de puertas mucho menor que 2^n), siendo programables las conexiones entre las entradas y las puertas Y (Figura 5.29b). A la salida se tiene una matriz de puertas O, estando fijas las conexiones entre las salidas de las puertas Y y las entradas de las puertas O.

Una **PLA** (***P****rogrammable* ***L****ogic* ***A****rray*) tiene de nuevo n entradas, una matriz de puertas Y (con un número de puertas mucho menor que 2^n), y una matriz de puertas O de salida, pero en este caso son programables tanto las conexiones de las entradas con las puertas Y como las de las puertas Y con las puertas O (Figura 5.29c).

Las matrices de puertas Y y de puertas O de las ROMs, PALs y PLAs, dependiendo de la tecnología de fabricación, pueden haberse sustituido por matrices de puertas NAND o por puertas NOR, pero estas posibles sustituciones no modifican la idea general de programabilidad.

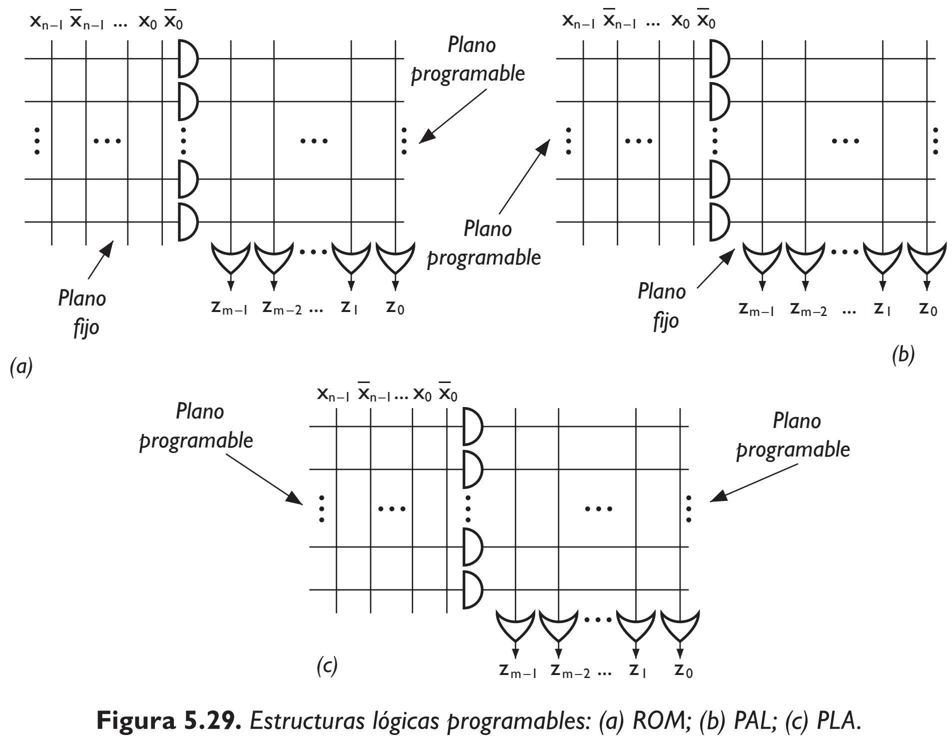


Figura 5.29. *Estructuras lógicas programables: (a) ROM; (b) PAL; (c) PLA.*

Además de estos chips programables, hay disponibles en el mercado otros tipos que incluyen también elementos secuenciales, de manera que con ellas se puede diseñar un sistema digital completo.

5.4.10 ALU

Una unidad aritmético-lógica (**ALU**: ***A****ritmetic and* ***L****ogic* ***U****nit*) es un sistema combinacional con $2n + p$ estradas de datos, m entradas de control, $n + p$ salidas de datos, y q salidas auxiliares, diseñado para poder realizar determinadas operaciones aritméticas o lógicas con las entradas de datos, según se decida mediante las entradas de control.

Dado que hay ALUs integradas fácilmente utilizables, lo que se va a hacer a continuación es describir uno de estos chips, el 74181, que tiene 24 patas, según se representa en la Figura 5.30. Tiene 9 entradas de datos ($n = 4$; $p = 1$), 5 entradas de control ($m =5$), 5 salidas de datos ($n = 4$; $p = 1$) y 3 salidas auxiliares ($q = 3$). Esta unidad puede realizar las operaciones aritméticas suma y resta, y como operaciones lógicas puede sintetizar las 16 funciones de conmutación de dos variables.

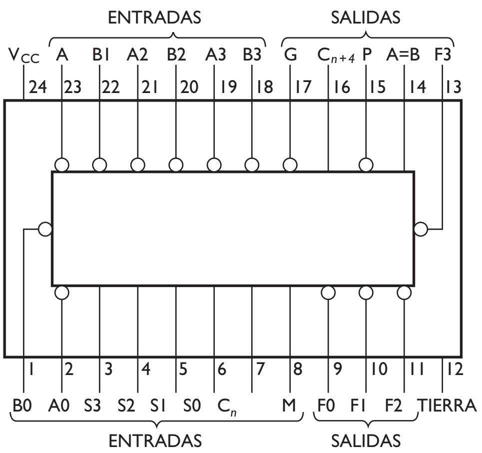


Figura 5.30. *Terminales del circuito integrado ALU 74181.*

Las operaciones aritméticas las realiza considerando que las nueve entradas de datos están estructuradas en dos valores o caracteres, **A** y **B**, de 4 bits cada uno, más una entrada de acarreo (o adeudo):

A: $A_3A_2A_1A_0$ } operandos
B: $B_3B_2B_1B_0$
$\mathbf{C_n}$**:** Acarreo o adeudo de entrada

Cuando se realiza una operación aritmética, las salidas de datos constan de un valor de 4 bits, que es el resultado de la operación aritmética correspondiente, más un bit de acarreo (o adeudo):

F: $F_3F_2F_1F_0$ Resultado
$\mathbf{C_{n+4}}$**:** Acarreo o adeudo de salida

Por tanto, como unidad aritmética, puede funcionar, por ejemplo, como un sumador de 4 bits, además de otras posibilidades de operaciones aritméticas.

Las operaciones lógicas las realiza operando bit a bit individualmente con los cuatro de cada una de las entradas. Por ejemplo, si se selecciona realizar la operación lógica O, las salidas F_i proporcionan los valores siguientes:

$$\mathbf{F_i} = A_i + B_i\ ;\ i = 0, \ldots, 3$$

Cuando se realizan operaciones lógicas, los bits de acarreo (de entrada y de salida) no tienen significado alguno.

De las tres salidas auxiliares, dos (G y P) se utilizan para poder unir en paralelo varios chips ALU y que el retardo global no crezca excesivamente (se genera un **acarreo anticipado**); en esta descripción simple no vamos a entrar en más detalles con respecto a esta posibilidad. La tercera salida auxiliar, "A = B", realiza una función de comparación e indica si los dos valores de entrada son o no iguales.

Las cinco entradas de control se agrupan en dos bloques, **M**, de un sólo bit, y **S** = $S_3S_2S_1S_0$, de cuatro. Con **M** se decide si la operación a realizar es aritmética (**M** = 0) o lógica (**M** = 1); con **S** se selecciona la operación concreta a realizar, dentro de cada grupo y según se especifica en la Tabla 5.3. En esta tabla se han indicado las operaciones lógicas Y y O con los símbolos · y +, respectivamente, y las operaciones aritmética suma y resta con las palabras "más" y "menos". En la columna correspondiente a las operaciones lógicas (**M** = 1), además de la expresión Y, O, NO de la función, se da, entre paréntesis, la función de conmutación de dos variables sintetizada para cada valor de **S**.

Tabla 5.3. *Operaciones realizadas por la ALU 74181.*

$S_3\ S_2\ S_1\ S_0$	**M = 0** **Operación aritmética**	**M = 1** **Operación lógica**	
0000	F = A más C_n	$F_i = \overline{A}_i$	(f_{12})
0001	F = (A + B) más C_n	$F_i = A_i + \overline{B}_i$	(f_{14})
0010	F = (A + $\overline{B}$) más C_n	$F_i = \overline{A}_i \cdot B_i$	(f_4)
0011	F = –1 más C_n	$F_i = 0$	(f_0)
0100	F = A más (A · $\overline{B}$) más C_n	$F_i = \overline{A}_i \cdot \overline{B}_i$	(f_8)
0101	F = (A + B) más (A · $\overline{B}$) más C_n	$F_i = \overline{B}_i$	(f_{10})
0110	F = A menos B menos 1 más C_n	$F_i = A_i \cdot \overline{B}_i + = \overline{A}_i \cdot B_i$	(f_6)
0111	F = (A · $\overline{B}$) menos 1 más C_n	$F_i = A_i \cdot \overline{B}_i$	(f_2)
1000	F = A más (A · B) más C_n	$F_i = \overline{A}_i \cdot B_i$	(f_{13})
1001	F = A más B más C_n	$F_i = A_i \cdot B_i + = \overline{A}_i \cdot \overline{B}_i$	(f_9)
1010	F = (A + $\overline{B}$) más (A · B) más C_n	$F_i = B_i$	(f_5)
1011	F = (A · B) menos 1 más C_i	$F_i = A_i \cdot B_i$	(f_1)
1100	F = A más A más C_n	$F_i = 1$	(f_{15})
1101	F = (A + B) más A más C_i	$F_i = A_i + \overline{B}_i$	(f_{11})
1110	F = (A + $\overline{B}$) más A más C_n	$F_i = A_i + B_i$	(f_7)
1111	F = A menos 1 más C_n	$F_i = A_i$	(f_3)

5.4.11 Realización de funciones combinacionales

Una vez obtenida la función de conmutación que se quiera sintetizar, hay varias opciones para su realización física. La elección entre estas opciones depende fundamentalmente del número de ejemplares que haya que construir del circuito a fabricar. Si lo que se desea es construir uno o muy pocos ejemplares del sistema, la opción es utilizar chips con puertas o bloques estándares, pues de esta forma se obtiene el diseño en menos tiempo y resulta más barato. Pero en este caso también se tienen varias posibilidades para sintetizar las funciones combinacionales concretas: utilizar puertas Y, O y NO; utilizar puertas NAND; utilizar puertas NOR; utilizar multiplexores; tratar de utilizar otros bloques estándares.

Si la producción ha de ser más voluminosa (digamos cientos o miles de copias del sistema que se está diseñando), entonces debe considerarse la posibilidad de utilizar en mayor o menor medida circuitos programables (ROMs, PALs o PLAs). También debe considerarse la posibilidad de diseñar un circuito VLSI específico para el sistema que se pretenda construir, dependiendo del volumen de producción al que se vaya a llegar.

5.5 Conclusión

En este capítulo se han visto los aspectos esenciales en el diseño de circuitos digitales combinacionales. También se han descrito los bloques digitales combinacionales para la realización de las operaciones más usuales (sumadores, comparadores, codificadores, decodificadores, multiplexores, demultiplexores, ALUs), así como los principales bloques programables (ROMs, PLAs, PALs). Conocer todo lo anterior es imprescindible para seguir los siguientes niveles descriptivos en el estudio de los computadores Además, también se tiene una capacitación suficiente para poder abordar el diseño de sistemas combinacionales de mediana complejidad, tanto a nivel de puertas lógicas como de elementos programables.

Como complemento a las descripciones de los circuitos y bloques combinacionales anteriormente realizadas es recomendable explorar las páginas web de los diferentes fabricantes de circuitos integrados, que se dan al final del capítulo, y analizar las características de cada uno de los circuitos disponibles en el mercado.

Este capítulo y el siguiente forman una unidad. Con ambos se aborda el diseño de sistemas digitales. Por tanto los conceptos aquí presentados se complementan con los que se presentarán en el capítulo siguiente.

EJERCICIOS

5.1 Utilizando los postulados del álgebra de Boole, demostrar que: para todo a, b, c $\in$ B,

$$ab + \overline{a}c + bc = ab + \overline{a}c$$

5.2 Utilizando los postulados y teoremas del álgebra de Boole, comprobar que

$$(a + \overline{b} + a \cdot b) \cdot (a + \overline{b}) \cdot \overline{a} \cdot b = 0$$

5.3 Utilizando reiteradamente las leyes de De Morgan, obtener una expresión en forma de suma de productos para las siguientes funciones:

a) $H(x,y,z,u,v,w) = \overline{(\overline{x \cdot \overline{y} + v \cdot w}) \cdot (\overline{\overline{x} \cdot \overline{z} + u \cdot \overline{v}})}$

b) $G(x,y,z) = \overline{(\overline{x + \overline{y} \cdot z}) \cdot (y + x \cdot z)}$

5.4 Suponiendo que se cumple que $x = \overline{y} \cdot z + y \cdot \overline{z}$, comprobar que se verifica que $z = \overline{x} \cdot y + \overline{x} \cdot \overline{y}$.

5.5 Demostrar que utilizando únicamente la función NAND se puede sintetizar cualquier función de conmutación. Ídem para la función NOR.

5.6 Utilizando el teorema de Shannon, desarrollar la siguiente función como suma de minterms y como producto de maxterms:

$$F = a \cdot b + (a + \overline{c}) \cdot (a + \overline{b})$$

5.7 Minimizar las siguientes funciones, dadas en la Figura 5.31 en forma de mapa de Karnaugh.

xy \ zu	00	01	11	10
00	_0_	_1_ 1	_3_ 1	_2_
01	_4_ 1	_5_	_7_	_6_ 1
11	_12_	_13_ 1	_15_ 1	_14_
10	_8_	_9_	_11_	_10_

xy \ zu	00	01	11	10
00	_0_ 1	_1_	_3_	_2_ 1
01	_4_	_5_ 1	_7_ 1	_6_
11	_12_	_13_ 1	_15_ 1	_14_
10	_8_ 1	_9_	_11_	_10_ 1

xy \ zu	00	01	11	10
00	_0_	_1_	_3_	_2_ 1
01	_4_	_5_ 1	_7_	_6_
11	_12_	_13_	_15_ 1	_14_
10	_8_ 1	_9_ 1	_11_	_10_ 1

Figura 5.31. *Mapas de Karnaugh.*

5.8 Utilizando el mapa de Karnaugh, minimizar las siguientes funciones en forma de suma de productos:

a) $f(X,Y,Z,W) = \Sigma m(0,2,3,6,8,11,12)$
b) $g(a,b,c,d) = \Sigma m(1,3,5,9,13)$
c) $F(x,y,z,u) = \Pi M(0,1,2,5,6,8,10,15)$
d) $G(A,B,C,D) = \Pi M(4,5,8,9,11,13)$

5.9 Utilizando el mapa de Karnaugh, minimizar las siguientes funciones en la forma de producto de sumas:

a) $f(X,Y,Z,W) = \Sigma m(1,2,3,4,6,8,11)$
b) $g(a,b,c,d) = \Sigma m(1,4,7,9,12)$
c) $F(x,y,z,u) = \Pi M(0,1,3,4,6,8,10,11,15)$
d) $G(A,B,C,D) = \Pi M(2,5,8,11,12,13)$

5.10 Utilizando el mapa de Karnaugh, minimizar las siguientes funciones en la forma de suma de productos:

a) $F(x,y,z,w) = y \cdot \overline{z} + x \cdot y \cdot w + x \cdot z \cdot w + x \cdot \overline{z} \cdot \overline{w}$
b) $G(x,y,z,v) = (x + \overline{y} + z) \cdot (\overline{x} + \overline{v}) \cdot (\overline{x} + y + v)$
c) $f(x,y,z,w) = x \cdot \overline{z} + y \cdot w + x \cdot z \cdot w + \overline{z} \cdot \overline{w}$
d) $h(x,y,z,v) = (x + \overline{y}) \cdot (\overline{x} + \overline{v}) \cdot (y + v)$

5.11 Utilizando el mapa de Karnaugh, minimizar las siguientes funciones en la forma de producto de sumas:

a) $F(x,y,z,w) = x \cdot \overline{y} \cdot \overline{z} + x \cdot y \cdot w + x \cdot \overline{z} \cdot w + x \cdot \overline{z} \cdot \overline{w}$
b) $h(x,y,z,v) = (x + \overline{y} + z)\ (\overline{x} + \overline{z} + \overline{v})(\overline{x} + \overline{y} + v)$
c) $f(x,y,z,u) = x \cdot \overline{y} \cdot u + \overline{x} \cdot z + y \cdot \overline{z} \cdot u$
d) $G(x,y,z,w) = y \cdot \overline{z} + x \cdot w + x \cdot z \cdot w + x \cdot \overline{w} \quad \overline{w}$

5.12 Representar en un mapa de Karnaugh la función F(x,y,z) dada en el cronograma de la Figura 5.4a; minimizar dicha función y obtener su expresión mínima en la forma de suma de productos; sintetizar esta expresión mínima utilizando sólo puertas NAND. Obtener la expresión mínima de F(x,y,z) en la forma de producto de sumas; sintetizar esta expresión mínima utilizando sólo puertas NOR.

5.13 Diseñar un detector de números primos con cinco entradas, E0 ..., E4, que representan en binario los números naturales menores que 32.

5.14 Un código que se utiliza con las cintas de papel perforado es el Flexowriter, en el que se utilizan 6 bits para codificar cada carácter y que se da en la Figura 5.32, en octal, para letras y números (se supone que los restantes caracteres no se utilizan). Diseñar un circuito con dos salidas que indique si el carácter correspondiente es una vocal, o es una consonante, o un número par o un número impar.

A	30	G	13	M	07	S	24	Y	25	4	64
B	23	H	05	N	06	T	01	Z	21	5	62
C	16	I	14	O	03	U	34	O	77	6	66
D	22	J	32	P	15	V	17	I	37	7	72
E	20	K	36	Q	35	W	31	2	74	8	60
F	26	L	11	R	12	X	27	3	70	9	33

Figura 5.32. *Código Flexowriter.*

5.15 Para el código Flexowriter del problema anterior, suponiendo que sólo se pueden recibir caracteres alfanuméricos, diseñar un circuito que detecte caracteres erróneos.

5.16 Diseñar un circuito que permita detectar (produciendo una salida al valor 1, por ejemplo), si el carácter ASCII de entrada es:
a) un carácter numérico;
b) una vocal;
c) un carácter de control.

5.17 Diseñar un circuito que genere un bit de paridad par para información BCD paralelo.

5.18 Diseñar un circuito que detecte paridad impar para información BCD paralelo.

5.19 Un circuito combinacional tiene cuatro entradas *a*, *b*, *c* y *d*, asociadas cada una de ellas a un conmutador, tomando el valor 1 si el conmutador está cerrado y el valor 0 si está abierto. La salida del circuito, *z*, debe ser 0 si, y sólo si se abren simultáneamente los conmutadores *c* y *a* o *b* y *d*. Obtener:
a) La tabla verdad del circuito.

b) Una función de conmutación que describa el comportamiento del circuito.
c) El circuito que sintetiza la función, realizado con circuitos Y, O y NO.

5.20 Diseñar un circuito que tenga como entradas dígitos BCD (habrá cuatro entradas binarias) y que su salida indique si el dígito de entrada es múltiplo de 3.

5.21 Diseñar un circuito restador binario de 1 bit.

5.22 Diseñar, utilizando cualquier elemento combinacional, un circuito que tenga como entradas dos números binarios de 2 bits: A = A1A0 y B = B1B0 y detecte, produciendo salida 1, si A > B.

5.23 Utilizando sumadores binarios, diseñar un sistema combinacional con 8 entradas, que se interpretan como un número binario sin signo. A la salida debe proporcionar el cociente entero que resulte al dividir la entrada por 5. Para resolverlo puede ser de ayuda tener en cuenta que $0{,}2_{10} = 0{,}00110011..._2$.

5.24 Utilizando comparadores y multiplexores, diseñar un sistema combinacional para ordenar cuatro números binarios de cuatro bits cada uno.

5.25 Diseñar un «codificador» que traslade información del código Gray al código BCD.

5.26 Un codificador de posición de un eje proporciona una señal de 4 bits que indica la posición del eje en incrementos de 30°. Utilizando el código que se da en la Tabla 5.4, diseñar un circuito lógico mínimo con una salida que detecte si la posición del eje corresponde al primer cuadrante (entre 0 y 90°):
a) Utilizando puertas Y, O y NO.
b) Utilizando multiplexores de *2 a 1* y de *4 a 1*, indistintamente.
c) Si se le ocurre alguna otra forma de realización, propóngala.
Se supone que las combinaciones de bits de salida del codificador que no aparecen en la tabla anterior, nunca se presentan y se pueden tomar como 0 o como 1, a conveniencia del diseñador.

Tabla 5.4.

Posición del eje	Salida del codificador			
	x	y	z	u
0-30º	0	0	1	1
30-60º	0	0	1	0
60-90º	0	1	1	0
90-120º	0	1	1	1
120-150º	0	1	0	1
150-180º	0	1	0	0
180-210º	1	1	0	0
210-240º	1	1	0	1
240-270º	1	1	1	1
270-300º	1	1	1	0
300-330º	1	0	1	0
330-360º	1	0	1	1

5.27 Diseñar un decodificador para el código Gray.

5.28 a) Diseñar un circuito combinacional con 4 entradas y 7 salidas. Las salidas van conectadas directamente a los siete segmentos de un módulo visualizador como el mostrado en las Figuras 9.27 y 11.8a. El circuito a sintetizar debe ser tal que se visualice el código hexadecimal de la combinación de entrada.
b) Con circuitos decodificadores como los anteriores,indicar el esquema de un sistema para visualizar en hexadecimal el contenido de una palabra de 32 bits.

5.29 Diseñar un demultiplexor de *1 a 8* utilizando demultiplexores de *1 a 2* y de *1 a 4*.

5.30 Sintetizar, utilizando multiplexores, las funciones de conmutación dadas en forma de mapas de Karnaugh en el Problema 5.7.

5.31 Diseñar una memoria ROM con la que se sintenticen las tres funciones del Problema 5.7.

5.32 Diseñar una **ROM** (es decir, dar las palabras de salida) que genere el cuadrado de un número entero dado en binario, para caracteres de 4 bits.

BIBLIOGRAFÍA

[Bae97] Baena, C.; Bellido, M. J.; Molina, A. J.; Parra, M. P.; Valencia, M: Problemas de circuitos y sistemas digitales, McGraw-Hill, 1997.

[Bro00] Brown, S.; Vranesic, Z.: *Fundamentals of Digital Logic with VHDL Design*, McGraw-Hill, 2000.

[Erc99] Ercegovac, M.; Lang, T.; Moreno, J. H.: *Introduction to Digital Systems*, John Wiley & Sons, 1999.

[Flo00] Floyd, T. L.: *Fundamentos de Sistemas Digitales*, Prentice Hall, 1997.

[Gaj97] Gajski, D. D.: *Principios de Diseño Digital*, Prentice Hall, 1997.

[Kat94] Katz, R. H.: *Contemporary Logic Design*, The Benjamin/Cummings Publishing Company, Inc, 1994.

[Lal96] Lala, P. K.: *Practical Digital Logic Design and Testing*, Prentice Hall, 1996.

[Llo03] Lloris, A., Prieto, A., Parrilla, L.: *Sistemas Digitales*, McGraw-Hill, 2003.

[Nel95] Nelson, V. P.; Nagle, H. T.; Carroll, B, D.; Irwin, D.: *Digital Logic Circuit Analysis & Desing*, Prentice Hall, 1995.

[Tin00] Tinder, R. F.: *Engineering Digital Desing*, Academic Press, 2000.

[Vin04] Vingron, S. P.: *Switching Theory*, Springer, 2004.

Web

Principales fabricantes de circuitos integrados:

http://www.actel.com
http://www.altera.com
http://www.atmel.com
http://www.amd.com
http://www.fairchildsemi.com
http://www.cypress.com
http://www.fairchildsemi.com
http://www.latticesemi.com
http://www.national.com
http://www.semiconductors.philips.com
http://www.xilinx.com

CAPÍTULO 6

Sistemas digitales secuenciales

En este capítulo se realiza una introducción al diseño de sistemas digitales secuenciales, presentando en primer lugar la distinción entre sistemas combinacionales y sistemas secuenciales. En segundo lugar se describen los diferentes elementos de memoria, componentes imprescindibles en el diseño de sistemas secuenciales. A continuación se analizan diferentes bloques secuenciales, como son los registros y los contadores, disponibles en forma de circuitos integrados y ampliamente utilizados en la construcción de cualquier sistema digital de mediana complejidad. Posteriormente se describen los diferentes tipos de memorias, tanto de acceso aleatorio (memorias RAM), como de acceso secuencial (memorias LIFO y FIFO). Con los Capítulos 5 y 6 se pretende dar una visión global del objetivo del diseño lógico y de los elementos constructivos con los que se materializa cualquier sistema digital, entre los que se encuentran los computadores.

6.1 Sistemas combinacionales y sistemas secuenciales

El capítulo anterior está dedicado a los sistemas digitales combinacionales. Se dice que un circuito o dispositivo digital, con entradas $x_1,...,x_n$, y salidas $z_1,...,z_m$, es combinacional si las salidas en un instante cualquiera, *t*, están determinadas exclusivamente por las entradas en ese mismo instante. Es decir:

$$z_i(t) = f_i(x_1(t),...,x_n(t)), \qquad i=1,...,m$$

Por tanto, la misma combinación de entradas (los mismos valores en las entradas) aplicada en instantes diferentes producirá siempre las mismas salidas.

Normalmente no se explicita la variable *t*, escribiéndose de forma más simple:

$$z_i = f_i(x_1,...,x_n), \qquad i=1,...,m$$

En esta definición se está suponiendo que el tiempo de propagación de las señales a través del circuito o dispositivo en cuestión es suficientemente pequeño, de manera que cualquier variación en el valor de las entradas afecta "instantáneamente" a las salidas.

Todos los ejemplos de sistemas digitales vistos en el Capítulo 5 corresponden a sistemas combinacionales y se ajustan a esta definición.

Un sistema es **secuencial** cuando las salidas en un instante cualquiera están determinadas por las entradas en ese instante y por la "historia" del sistema. Es decir, un sistema secuencial

puede producir salidas diferentes para las mismas combinaciones de entradas si éstas se aplican en instantes diferentes. La "historia" del sistema está determinada por el estado inicial en el que empieza a funcionar el sistema y por las entradas que ha recibido; esta historia se cuantifica, se concreta, en diferentes **estados**, de manera que, en términos generales, el sistema empieza a funcionar en un estado inicial y evoluciona pasando de un estado a otro, según sean las entradas en cada instante. En este caso se dice que la salida en cualquier instante es función de las entradas en ese instante y del estado en el que se encuentre el sistema. Para representar el estado se utilizan variables binarias de estado.

Un ejemplo de sistema secuencial podría ser el sistema de control de un ascensor. Las entradas al sistema son las pulsaciones en los botones del ascensor. La respuesta del sistema consiste en subir o bajar una determinada altura, y esta respuesta depende de la entrada, pero también del "estado" en el que se encuentre el sistema (en este caso, del piso en el que esté). Si el ascensor está en la planta 1 y se le pulsa al botón 2, subirá la altura de una planta; pero si está en la planta 4 y se le pulsa al botón 2, bajará la altura de dos plantas. Por tanto, la misma entrada (pulsar al botón 2) produce dos salidas diferentes correspondientes a dos estados diferentes.

Los sistemas combinacionales no "recuerdan" cuál ha sido su historia, y se dice que son **sistemas sin memoria**. Los sistemas secuenciales recuerdan, hasta un determinado nivel, su historia previa, y se dice que son **sistemas con memoria**. Para diseñar sistemas secuenciales serán necesarios unos nuevos componentes con los que poder "recordar" la historia del sistema, los **elementos de memoria**, a los que se dedica el apartado siguiente.

6.2 Elementos de memoria

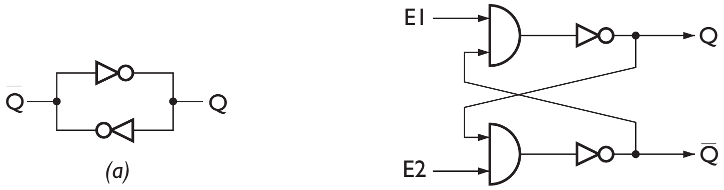
Para diseñar sistemas secuenciales es preciso utilizar algún dispositivo que sea capaz de "recordar" o "almacenar" información que previamente se le haya introducido. Una forma de cubrir este objetivo consiste en utilizar los elementos de memoria que a continuación vamos a describir.

El elemento de memoria básico, o almacén de un bit, se obtiene acoplando dos circuitos inversores de manera que la salida de uno sea la entrada del otro, y viceversa, tal como se representa en la Figura 6.1a. Es fácil comprobar que esta asociación de dos inversores, a la que llamaremos **biestable** o **flip-flop**, puede estar en uno de dos estados estables: $Q = 1$ ($\overline{Q} = 0$) ó bien $Q = 0$ ($\overline{Q} = 1$), conocidos como estados 1 y 0, respectivamente. Actuando a través de los terminales Q o $\overline{Q}$ se puede "escribir" en el biestable cualquier información que se desee, entendiendo por escribir el situarlo en el estado 1 ó 0; la información escrita en el biestable permanece en él mientras esté adecuadamente conectado a la tensión de alimentación. Esta información almacenada por el biestable puede leerse en los puntos Q y $\overline{Q}$ cuantas veces se desee.

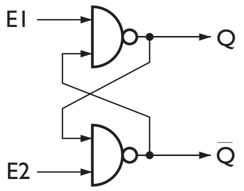
Para facilitar los procesos de lectura y escritura y dar mayor versatilidad a los elementos de memoria, resulta adecuado añadirle dos entradas (E_1 y E_2) al biestable de la Figura 6.1a que, junto con las entradas de realimentación, pasen a través de sendas puertas Y antes de actuar como entradas de los inversores. El circuito resultante se representa en la Figura 6.1b. Cada puerta Y seguida de un inversor es una puerta NO-Y o NAND, y el circuito de la Figura 6.1b se representará como en la Figura 6.1c.

Para facilitar el análisis del funcionamiento de este dispositivo, en la Figura 6.1d se reproduce la tabla verdad de la función NAND; de ella se concluye que cuando una de las dos

entradas de una puerta NAND está al valor 1, la salida de la puerta NAND es el complemento de la otra entrada (es decir, en este caso la puerta NAND actúa como inversor).



(a) *(b)*



(c)

x	y	NAND
0	0	1
0	1	1
1	0	1
1	1	0

(d)

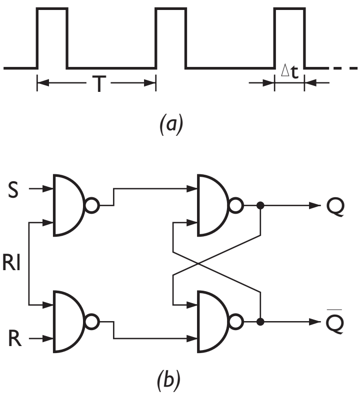
Figura 6.1. *(a) Elemento de memoria formado por dos inversores; (b) elemento de memoria incluyendo puertas Y para las entradas; (c) representación del anterior utilizando puertas NAND; (d) tabla verdad de la función NAND.*

Vamos a analizar a continuación el dispositivo de la Figura 6.1c. Cuando $E1 = 0$ y $E2 = 0$, resultan $Q = 1$ y $\overline{Q} = 1$; en este caso no se cumple que las dos salidas sean complementarias; por esta razón y por otra que se verá posteriormente, se considera que esta combinación de entradas debe evitarse. Cuando $E1 = 0$ y $E2 = 1$, es $Q = 1$ y $\overline{Q} = 0$; esta combinación de entradas, por consiguiente, sirve para poner a 1 el elemento de memoria. Cuando $E1 = 1$ y $E2 = 0$, resulta $Q = 0$ y $\overline{Q} = 1$; con esta combinación de entradas se pone a 0 el elemento de memoria. Cuando se aplican $E1 = 1$ y $E2 = 1$, las salidas Q y $\overline{Q}$ permanecen al valor que tuviesen previamente, como es fácil comprobar; esta combinación de entradas es la combinación de reposo y sirve para mantener inalterada (memorizada) la información que previamente se hubiese almacenado. Por tanto, con *E1* y *E2* se puede escribir en el biestable cualquier información (un 0 ó un 1) y se puede controlar que esa información se mantenga indefinidamente en el mismo.

6.2.1 Biestables disparados por nivel

Es frecuente que el funcionamiento de los sistemas secuenciales esté controlado por una señal de reloj, *Rl*, de manera que las transiciones en las variables de estado tengan lugar únicamente cuando la señal de reloj valga $Rl = 1$, y cuando sea $Rl = 0$ el sistema permanezca en el mismo estado. La señal de reloj es una señal periódica que cada T segundos toma el valor 1, permaneciendo en este valor durante Δt segundos, tal como se representa en la Figura 6.2a. Concretando a los elementos de memoria, cuando sea $Rl = 0$, debe ocurrir que $E1 = E2 = 1$, para que el biestable permanezca con su valor anterior. Cuando sea $Rl = 1$, con las señales externas (que se denominarán S y R) se situará al elemento de memoria en el estado que se desee. El lector puede comprobar que con el circuito de la Figura 6.2b se consigue el funcionamiento síncrono que se desea. Este circuito deriva del de la Figura 6.1c, al que se le ha añadido otro nivel de puertas NAND, con dos entradas de control externas,

S (de *Set*: poner a uno) y **R** (de *Reset*: poner a cero), y una entrada de reloj, *Rl*. Sólo cuando $Rl = 1$ se puede modificar el estado de este biestable, que se conoce como **biestable SR**; cuando $Rl = 0$, la salida no varía. Cuando el reloj está activo (al valor 1), con $SR = 00$ resulta que la salida en el instante $n + 1$ coincide con la salida en el instante n ($Q_{n+1} = Q_n$); con $SR = 01$ se obtiene $Q_{n+1} = 0$; con $SR = 10$ queda $Q_{n+1} = 1$; con $SR = 11$ resulta $Q_{n+1} = 1$ y también $\overline{Q}_{n+1} = 1$. Ésta es la situación a evitar mencionada anteriormente, y la razón adicional a la que se aludió es la siguiente: cuando $Rl = 1$ y $SR = 11$, las salidas toman el valor $Q = 1$ y $\overline{Q} = 1$, pero ambas no pueden mantenerse a este valor 1 cuando la señal de reloj pase a valer 0, en cuyo caso, debido a que los retardos en la propagación de las señales a través de los diferentes caminos son distintos, es impredecible el estado en que quedará el biestable (lo que sí es claro es que una salida quedará a cero y la otra a uno); es decir, para $SR = 11$ se tiene una situación de indeterminación cuando la señal de reloj pasa a valer $Rl = 0$.



Q_n	S	R	Q_{n+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	*
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	*

(c)

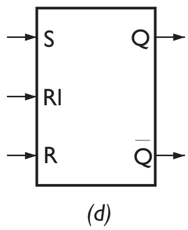


Figura 6.2. *(a) Pulsos de reloj; (b) biestable SR con reloj; (c) tabla del biestable SR; (d) representación del biestable SR.*

En la tabla de la Figura 6.2c se tiene descrito el funcionamiento del biestable *SR*, expresando Q_{n+1} como función de las entradas *SR* y de la salida anterior Q_n; las dos situaciones de indeterminación se han representado mediante un símbolo *. Un biestable *SR* se representará como en la Figura 6.2d.

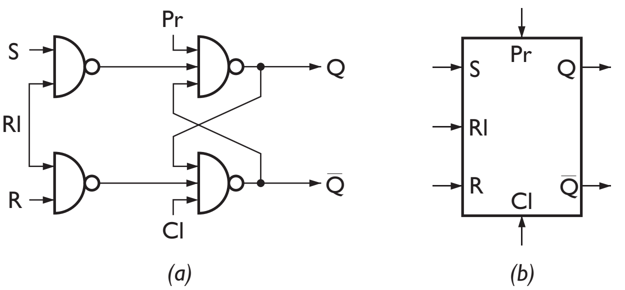


Figura 6.3. *Biestable SR con Preset y Clear: (a) circuito; (b) representación.*

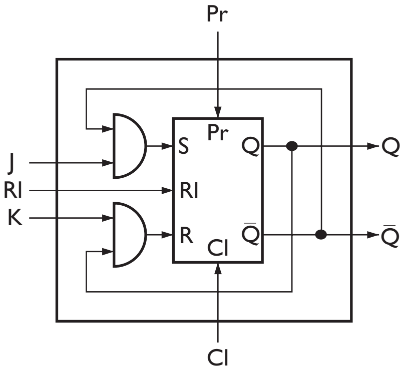
En ocasiones los biestables *SR* disponen de dos entradas más, conocidas como **entradas asíncronas**, denotadas respectivamente por *Pr* (de *Preset*: fijar a uno) y *Cl* (de *Clear*: borrar o poner a cero). Con estas entradas asíncronas se puede poner el biestable al valor cero o

al uno con independencia de la señal de reloj. En el circuito de la Figura 6.3a se ve cómo incluir estas dos entradas asíncronas. Cuando se quiera actuar con las entradas síncronas deben mantenerse las entradas asíncronas a los valores $Pr = Cl = 1$. Para poner a cero con las entradas asíncronas hay que hacer $Pr = 1$ y $Cl = 0$; para poner a uno debe hacerse $Pr = 0$ y $Cl = 1$. Un biestable *SR* con *Preset* y *Clear* se representará como en la Figura 6.3b. En el mercado hay biestables que incluyen ambas entradas asíncronas; o bien sólo una (o ninguna) de las dos, bien *Preset*, bien *Clear*.

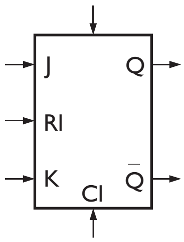
Un claro inconveniente del biestable *SR* es la indeterminación en las salidas cuando $SR = 11$. Este inconveniente se soluciona con el **biestable JK**, que funciona en todo igual al *SR* salvo que cuando $JK = 11$, el biestable cambia de estado ($Q_{n+1} = \overline{Q}_n$). Es decir, la tabla del biestable *JK* es la de la Figura 6.4a. Es fácil comprobar que a partir de un biestable *SR* se puede construir un biestable *JK* añadiéndole dos puertas Y, tal como se representa en la Figura 6.4b. El biestable *JK* se simboliza como en la Figura 6.4c. Es claro que el biestable *JK* puede tener también las entradas asíncronas *Pr* y *Cl*.

Q_n	J	K	Q_{n+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

(a)



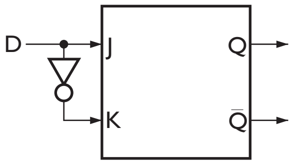
(b)



(c)

Figura 6.4. *Biestable JK: (a) tabla; (b) esquema del mismo a partir del biestable SR; (c) representación.*

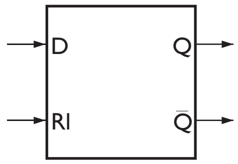
Otro biestable muy utilizado es el **biestable tipo D** (de *Delay*: retardo), que, además de la entrada de reloj, tiene una sola entrada, *D*, y funciona de manera que la salida Q_{n+1} en cualquier instante es la entrada D_n en el instante anterior; es decir, el biestable *D* es el que con más claridad actúa como elemento de almacenamiento, pues cualquier información que se aplique a la entrada se escribe en él. También es claro que el biestable *D* es un elemento de retardo de un periodo de reloj: la información que se aplica a la entrada aparece en la salida en el siguiente pulso de reloj.



(a)

D	Q_{n+1}
0	0
1	1

(b)



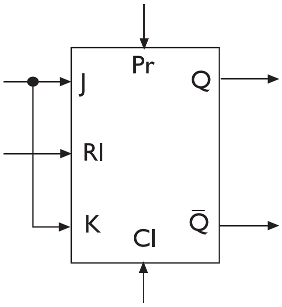
(c)

Figura 6.5. *Biestable D: (a) síntesis a partir del biestable JK; (b) tabla; (c) representación.*

A partir de un biestable *JK* (o de un *SR*) se puede construir un biestable *D* imponiendo que $K = \overline{J}$ (o $R = \overline{S}$), tal como se representa en la Figura 6.5a; con esta restricción en las entradas se reducen a cuatro las filas de la tabla de la Figura 6.4a: aquellas en las que *J* es distinta de *K*. En estas cuatro filas es $Q_{n+1} = J = D$. La tabla del biestable *D* es la de la Figura 6.5b y su representación la de la Figura 6.5c.

El último tipo de biestable que vamos a considerar es el **biestable tipo T**. Tiene una sola entrada síncrona, *T*, y funciona de manera que cuando $T = 1$, el biestable cambia de estado con cada pulso de reloj, y cuando $T = 0$ no cambia de estado.

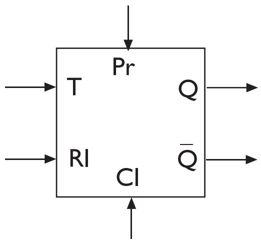
Directamente se comprueba que un biestable *JK* se puede transformar en un biestable *T* si se hace $J = K$, pues en estos casos siempre es $Q_{n+1} = \overline{Q}_n$ (es decir, cambia la salida) cuando $J = K = 1$, mientras que para $K = J = 0$ es $Q_{n+1} = Q_n$ (no cambia la salida). En la Figura 6.6a se representa esta forma de realización. La tabla del biestable *T* se da en la Figura 6.6b y su representación gráfica está en la Figura 6.6c.



(a)

Q_n	T	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	1

(b)



(c)

Figura 6.6. *Biestable T: (a) realización a partir del biestable JK; (b) tabla; (c) representación.*

En el proceso de síntesis resulta necesario conocer qué combinaciones de entradas deben aplicarse al biestable correspondiente para conseguir una determinada transición en las salidas. Esta información se puede obtener directamente de las tablas de los diferentes tipos de biestables (Figuras 6.2c, 6.4a, 6.5b y 6.6b). En la Figura 6.7 se dan en forma de tabla las combinaciones de entradas que se deben aplicar para conseguir las diferentes transiciones en las salidas. En estas tablas se utiliza el símbolo "**-**" para indicar que el valor de esa variable puede ser cualquiera, que es indiferente (se puede tomar como cero o como uno, según interese).

Las entradas síncronas de los biestables *SR, JK, D* y *T* previamente descritos están controladas por la señal de reloj. Concretamente para poder escribir en ellos cualquier valor utilizando las entradas síncronas ha de ser $Rl = 1$, y mientras esté a 1 la entrada de reloj, captan cualquier cambio en las entradas. Es decir, es el nivel 1 de la señal de reloj el que determina el funcionamiento de los biestables, que por esta razón se conocen como **biestables disparados por nivel.** Cuando se utilizan estos biestables, se supone que las entradas (*SR, JK, D* o *T*) cambian con $Rl = 0$ y permanecen constantes con $Rl = 1$, de manera que las transiciones en las salidas de los biestables están perfectamente definidas y, con ello, se garantiza un correcto funcionamiento. Cuando esta forma de actuación no se puede garantizar, los biestables disparados por nivel pueden presentar problemas, en cuyo caso conviene utilizar los biestables disparados por flanco o los biestables dueño-esclavo, que se describen a continuación.

Q_n	Q_{n+1}	S	R
0	0	0	–
0	1	1	0
1	0	0	1
1	1	–	0

(a)

Q_n	Q_{n+1}	J	K
0	0	0	–
0	1	1	–
1	0	–	1
1	1	–	0

(b)

Q_n	Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

(c)

Q_n	Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

(d)

Figura 6.7. *Tablas de excitación para los diferentes biestables: (a) SR; (b) JK; (c) D; (d) T.*

6.2.2 Biestables dueño-esclavo y disparados por flanco

Un biestable **dueño-esclavo** está formado por dos biestable disparados por nivel, dispuestos de manera que uno (**dueño**) está controlado por la señal de reloj y el otro (**esclavo**) por la señal de reloj complementada, y la salida del primero es la entrada del segundo, tal como se representa en la Figura 6.8 para el bistable *SR* dueño-esclavo. Nótese que el biestable esclavo está actuando como un biestable *D*, pues sus entradas siempre son mutuamente complementarias. Las entradas externas actúan sobre el dueño, y la salida del biestable *dueño-esclavo* se toma del esclavo.

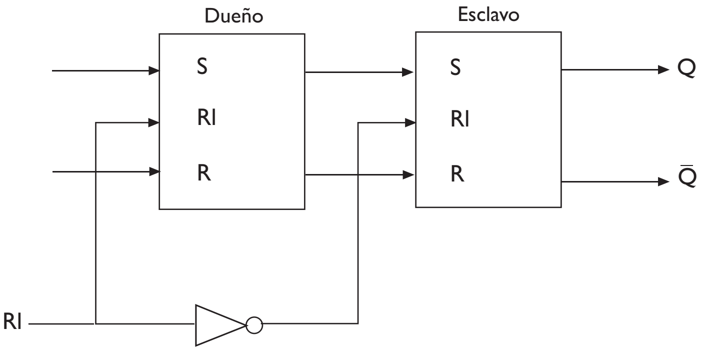


Figura 6.8. *Biestable SR dueño-esclavo.*

El biestable de la Figura 6.8 funciona de forma que cuando *Rl* pasa a valer 1, el dueño se sitúa en el valor determinado por las entradas *SR*, y cuando *Rl* pasa a valer 0, lo que esté escrito en el dueño se copia en el esclavo. Como se ve, se escribe en el dueño al inicio del pulso de reloj, y en el esclavo al final del pulso de reloj; por esta razón, a veces se dice que los biestables *dueño-esclavo* están **disparados por pulsos**. No obstante, cualquier cambio en las entradas externas cuando $Rl = 1$ es captado por el dueño.

Con idéntica estructura a la del biestable *SR* dueño-esclavo de la Figura 6.8, y con similar funcionamiento, hay biestables *JK, T* y *D* dueño-esclavo. Las tablas de excitación para los biestables dueño-esclavo son las mismas que las de la Figura 6.7.

Una característica del biestable dueño-esclavo es que la información contenida en el mismo (que está en el esclavo) no cambia hasta que Rl=0. Por tanto, con Rl=1 se puede estar escribiendo una información y simultáneamente estar leyendo la información previa, pues la escritura se hace sobre el dueño y la lectura del esclavo. Es decir, los biestables *dueño-esclavo* permiten la lectura y escritura simultaneas, lo que resulta muy útil en determinadas aplicaciones.

Para aislar los biestables de posibles cambios en las entradas cuando $Rl = 1$ se han construido los biestables disparados por flanco (tipo *SR, JK, T* o *D*), en cuya descripción no vamos a entrar en este curso de introducción. El cambio de 0 a 1 en la señal de reloj se conoce como **flanco de subida**, y el cambio de 1 a 0 es el **flanco de bajada**. En un biestable disparado por flanco, las entradas son relevantes únicamente en el flanco correspondiente, de subida si se trata de un **biestable disparado por flanco de subida**, o de bajada si es un **biestable disparado por flanco de bajada**. En cualquier caso, los cambios que pueda haber en las entradas siendo $Rl = 1$, fuera del punto concreto de disparo (el flanco de subida o el de bajada), no afectan al funcionamiento del biestable. Las tablas de excitación siguen siendo las de la Figura 6.7.

Hay disponibles múltiple chips con biestables *JK* o *D* con diferentes prestaciones. Por ejemplo, el chip 7473 (14 patas) incluye dos biestables *JK* independientes disparados por flanco de bajada, cada uno con entrada de reloj y entrada asíncrona *Cl*; el chip 74112 (16 patas) tiene dos biestables *JK* independientes disparados por flanco de bajada con reloj y entradas *Cl* y *Pr*; el chip 7474 (14 patas) incluye dos biestables *D* independientes disparados por flanco de subida con reloj y entradas *Cl* y *Pr*; el chip 74273 (20 patas) incluye ocho biestables *D* disparados por flanco de subida, que tienen la misma entrada de reloj y de *Cl*. Algunos chips incluyen biestables *SR*, aunque no es lo usual, y los biestables *T* no están disponibles en forma integrada (si interesa utilizarlos, se sintetizan a partir de los *JK*, tal como se ha visto anteriormente).

6.3 Registros

La capacidad de almacenamiento de un biestable es la mínima posible: un solo bit. En la práctica, en la mayoría de las ocasiones, la información se estructura y procesa en bloques de bits, denominados **palabras**, por lo que es necesario utilizar elementos de almacenamiento de palabras. Para esta finalidad se usan los registros.

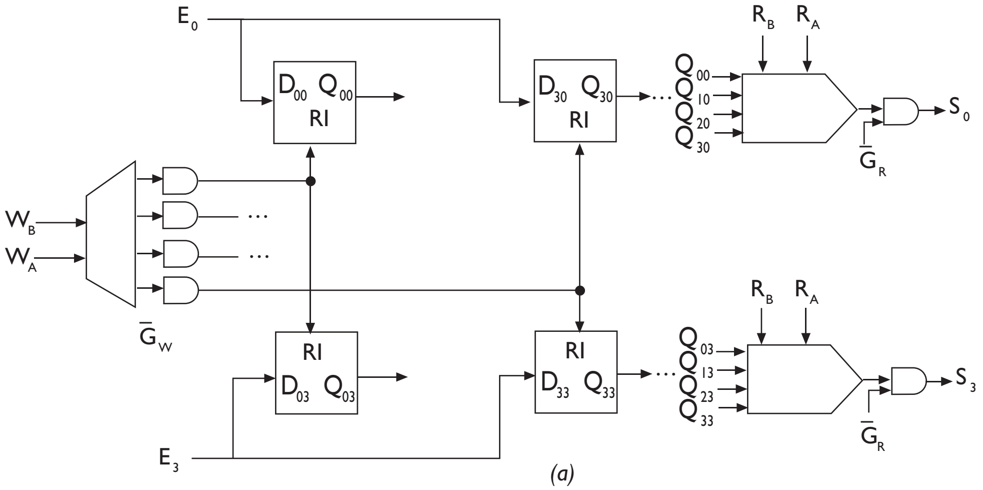
Un **registro** de *p* posiciones es un elemento de almacenamiento capaz de memorizar *p* bits. Un registro elemental que sólo permita leer o escribir la información contenida en el mismo está formado por *p* biestables independientes. Normalmente un registro permite realizar diferentes operaciones con la información contenida en el mismo, aparte de "leerla" o "escribirla"; en este caso habrá, en general, una relación funcional entre las entradas y las salidas de los diferentes biestables. En este apartado vamos a describir los bancos de registros, así como uno de los registros más utilizados, el registro de desplazamiento (*shift register* en inglés).

6.3.1 Bancos de registros

Un banco de registros está formado por un conjunto de $2^n x m$ biestables *dueño-esclavo* agrupados en 2^n palabras de *m* bits cada una. Estos biestables permiten la escritura y la lectura de información; por tanto en un banco de registros hay *m* entradas de datos, $E_{m-1},...,E_0$, para escribir la información; *m* salidas de datos, $S_{m-1},...,S_0$, para leer la información; *n* entradas de dirección para seleccionar la palabra en la que se va a escribir la información, $W_{n-1},...,W_0$; y otras *n* entradas de dirección para seleccionar la palabra de la que se va a leer la información, $R_{n-1},...,R_0$. Además, el banco de registros incluirá algunas otras entradas de control para decidir cuándo se va a leer o escribir una determinada información. Por ejemplo, en la Figura 6.9a se representa un banco de registros de cuatro palabras de cuatro bits, que corresponde al chip 74670, con salidas triestado; tiene cuatro entradas de datos, $E_3,...,E_0$, cuatro salidas

de datos, S_3, ..., S_0, dos entradas de dirección para seleccionar la palabra en la que se va a escribir la información, W_B y W_A, y otras dos entradas de dirección para seleccionar la palabra de la que se va a leer la información, R_B y R_A. Además tiene una entrada de control, G_W, con la que se decide si se escribe o no, y otra entrada de control, G_R, con la que se decide si la salida está en alta o en baja impedancia. En la Figura 6.9b se da la tabla de funcionamiento de este banco de registros.

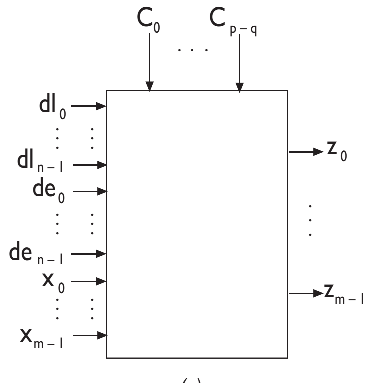
La representación esquemática que se utilizará para un banco de registros será la de la Figura 6.9c.



(a)

G_W W_B W_A	Palabra escrita	G_R R_B R_A	Palabra leída
0 0 0	0	0 0 0	0
0 0 1	1	0 0 1	1
0 1 0	2	0 1 0	2
0 1 1	3	0 1 1	3
1 – –	Ninguna	1 – –	Ninguna

(b)



(c)

Figura 6.9. *Bancos de registros: (a) de cuatro palabras de cuatro bits; (b) tabla de funcionamiento; (c) representación esquemática.*

El funcionamiento de un banco de registros queda descrito como sigue. Para escribir una palabra, en las entradas de datos, E_{m-1},...,E_0, se aplican los valores 0 ó 1 que se deseen introducir, las entradas de dirección W_{n-1},...,W_0 se ponen al valor adecuado para seleccionar la palabra en la que se quiere escribir, y las entradas de control se sitúan en los valores que correspondan a la operación de escritura. Por ejemplo, para escribir 0110 en la palabra 10 del banco de registros de la Figura 6.9a hay que hacer E_3 E_2 E_1 E_0= 0110, W_BW_A = 10, y G_W se ha de poner a 0. Es evidente que cuando en una palabra del banco de registros se escribe

una información, se destruye la información anterior; se dice que la operación de escritura es destructiva. Para leer la información contenida en una determinada palabra, en el caso general, basta aplicar en $R_{n-1},...,R_0$ la dirección correspondiente y en $S_{m-1},...,S_0$ se tendrán los valores contenidos en esas posiciones de memoria una vez que G_W se ponga a 0. Es claro que una misma información puede leerse cuantas veces se quiera, pues el proceso de lectura no modifica los contenidos de los biestables; se dice que la lectura es no destructiva.

Dado que hay entradas de dirección independientes para lectura y para escritura, se pueden simultanear ambas operaciones. Incluso se puede leer una palabra al mismo tiempo que se está escribiendo en ella; en este caso se escribe una nueva información mientras se lee la información previa.

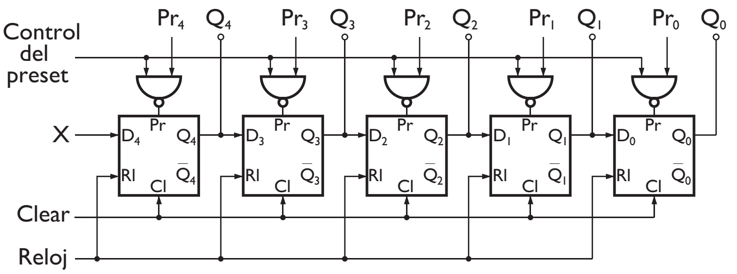
En lo que se ha dicho hasta ahora se ha supuesto que hay una sola salida de datos. Pero no hay inconveniente alguno para que hayan dos o más salidas de datos independientes: basta replicar cuantas veces se quiera los multiplexores de salida y añadir las entradas de control correspondientes.

6.3.2 Registros de desplazamiento

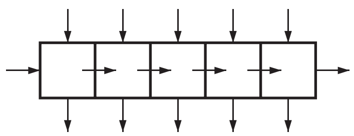
Un **registro de desplazamiento** de p bits es un sistema secuencial síncrono con una entrada externa, x, formado por p biestables conectados en cascada y cuyas salidas en el instante $n + 1$, $Q_{j,n+1}$, cumplen la siguiente relación:

$$Q_{j+1,n+1} = Q_{j,n}$$

$$Q_{1,n+1} = x_n$$



(a)



(b)

T	x	Q₄	Q₃	Q₂	Q₁	Q₀
1	1	1	–	–	–	–
2	1	1	1	–	–	–
3	0	0	1	1	–	–
4	1	1	0	1	1	–
5	0	0	1	0	1	1

(c)

Figura 6.10. *(a) Registro de desplazamiento de 5 bits; (b) representación esquemática de un registro de desplazamiento; (c) cronograma.*

Hay diferentes tipos de registros de desplazamiento, con varios grados de complejidad. Para ser más concretos en la descripción, en lo que sigue vamos a referirnos a un registro de desplazamiento de una complejidad media, comercialmente disponible, que está representado en la Figura 6.10a. Esquemáticamente se representará un registro de desplazamiento como en la Figura 6.10b.

El registro de desplazamiento de la Figura 6.10a, que corresponde al chip 7496, está constituido por cinco biestables *D* dueño-esclavo conectados en cascada y con entradas asíncronas. Las entradas *Clear* se activan conjuntamente mientras que las de *Preset* pueden actuar por separado, con una señal común de control. Hay una entrada serie síncrona, *x*, y todas las salidas Q_i están disponibles externamente. Vamos a suponer que por la línea *x* entra la información serie 01011, que se quiere escribir en el registro. (El convenio que se sigue usualmente en la transmisión serie es que el primer bit en llegar es el menos significativo). Apoyándonos en el cronograma de la Figura 6.10c, vamos a ver que aplicando a la entrada de reloj cinco pulsos al mismo tiempo que por la entrada *x* se introducen los cinco bits 01011, en el orden 1, 1, 0, 1, 0 (del menos significativo al mas significativo), se escribe la información en el registro. En efecto, cuando llegue el primer pulso será $x = 1$, de manera que esta información se escribirá en el biestable 4, resultando $Q_{4,1} = 1$. En $t = 2$ es de nuevo $x = 1$ (segundo bit a escribir), y la entrada del biestable 3 es también 1, resultando $Q_{4,2} = 1$, y $Q_{3,2} = 1$. En $t = 3$ es $x = 0$, quedando $Q_{4,3} = 0$, $Q_{3,3} = 1$, $Q_{2,3} = 1$. Y así sucesivamente hasta el quinto pulso de reloj, en el que quedarán $Q_{4,5} = 0$, $Q_{3,5} = 1$, $Q_{2,5} = 0$, $Q_{1,5} = 1$, $Q_{0,5} = 1$. Es decir, cada vez que llega un pulso, la información almacenada en el registro se desplaza una posición hacia la derecha; por ello este dispositivo se denomina **registro de desplazamiento hacia la derecha**. Una vez escrita esta información serie, si se quiere que permanezca en el registro, hay que detener la llegada de pulsos de reloj.

La información contenida en el registro es totalmente accesible en cualquier instante, pues todas las salidas Q_i son accesibles externamente; esto quiere decir que la información contenida en el registro, que se había escrito en serie, se puede leer en paralelo. En otras palabras, este registro de desplazamiento transforma una información serie, que se había introducido en cinco pulsos de reloj, en información en paralelo, que se lee en cinco puntos diferentes: pasa de un código temporal (cada bit aparece en un instante diferente) a un código espacial (cada bit ocupa una posición diferente).

La información almacenada en el registro también se puede leer en serie, a través de la salida Q_0. Para esto basta introducir cinco pulsos por la entrada de reloj. Esta operación de lectura en serie se podría simultanear con la escritura en serie. Si la señal de reloj no se detiene, el registro de desplazamiento con entrada serie y salida serie actúa como un retardo de *n* pulsos de reloj. También podrían realizarse las operaciones de lectura y escritura a ritmos diferentes, utilizando una señal de reloj Rl_e para la escritura y otra diferente, Rl_l, para la lectura; de esta forma se podría utilizar un registro de desplazamiento para trasvasar información entre dos sistemas que utilicen distintas señales de reloj.

Utilizando las entradas asíncronas, también se puede escribir en paralelo cualquier información que se desee. Esta operación de escritura se realiza en dos fases: en la primera, con la entrada de *Clear*, se ponen todos los biestables a cero; en la segunda, con las entradas de *Preset*, que se activan individualmente, se ponen a 1 aquellos biestables que deban almacenar este valor. Esta información escrita en paralelo puede leerse a continuación en serie o en paralelo; concretamente, si se escribe en paralelo y se lee en serie, este registro actúa como un conversor de paralelo a serie.

En resumen, hay cuatro modos de utilización del registro de desplazamiento: entrada serie-salida serie; entrada serie-salida paralelo (conversor serie-paralelo); entrada paralelo-salida serie (conversor paralelo-serie); y entrada paralelo-salida paralelo. En este último modo el registro actúa como un simple dispositivo de almacenamiento formado por cinco biestables independientes.

El registro de desplazamiento que hemos visto es de cinco bits, pero al ser accesibles todas sus salidas, también es un registro de desplazamiento de cualquier número de bits menor que cinco. Para obtener un registro de desplazamiento con mayor número de bits basta enlazar dos o más registros de desplazamiento como el descrito, de manera que la entrada x de cada uno (menos el primero) se conecte a la salida Q_0 del anterior.

Comercialmente están disponibles registros de desplazamiento que, mediante una señal de control o utilizando dos señales de reloj diferentes no simultaneas, permiten desplazamientos hacia la derecha (como el que se ha visto) o hacia la izquierda. Por ejemplo, el circuito 74195 (16 patas) es un registro de desplazamiento de 4 bits que permite la carga síncrona en paralelo, el desplazamiento a la derecha, el desplazamiento a la izquierda y la puesta a cero. También hay registros de desplazamiento más sencillos, que sólo permiten entrada serie y salida serie; o también entrada serie y salida serie o paralelo. Por ejemplo, el chip 74165 (16 patas) es un registro de desplazamiento con entrada en paralelo y salida en serie (por supuesto, también es posible la entrada en serie); el chip 74164 (14 patas) es un registro de desplazamiento de 8 bits con entrada en serie y salida en paralelo; con el 4006 (14 patas) se puede obtener un registro de desplazamiento de 18 bits con entrada serie y salida serie.

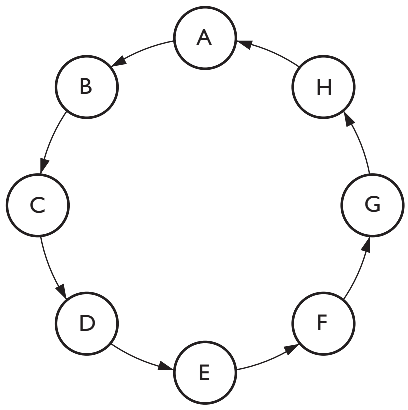
6.4 Contadores

Un **contador** es un sistema secuencial con una entrada de reloj (o, en general, con una entrada de pulsos) y unas salidas $z_0,...,z_m$, tal que las salidas presentan una serie de configuraciones preestablecidas y en un determinado orden, pasando de una configuración a la siguiente cada vez que llega un pulso de reloj. Los contadores más simples son los binarios, y a estos nos vamos a referir en lo que sigue.

Un **contador binario** módulo 2^n presenta a la salida, de forma cíclica, todas las configuraciones de n variables, en orden creciente si se trata de un **contador ascendente**, o en orden decreciente si se trata de un **contador descendente**. Por ejemplo, un contador binario ascendente módulo 8 presenta a la salida, sucesivamente, las siguientes configuraciones: 000, 001, 010, 011, 100, 101, 110, 111, 000, 001,... Como se ve, este dispositivo "cuenta" en binario, módulo 8, el número de pulsos que llegan; de ahí la denominación. Los pulsos de entrada no han de ser periódicos necesariamente, y actúan en el contador a través de la entrada de reloj.

Para ilustrar el procedimiento de diseño de este tipo de sistemas, se va a diseñar a continuación un contador binario ascendente módulo 8. Dado que hay ocho configuraciones diferentes de salidas, el sistema tiene ocho **estados**, que denominaremos *A, B, C, D, E, F, G*, y *H*, de manera que en el estado *A* la salida debe ser 000, en el *B*, 001, y así sucesivamente. Para codificar estos ocho estados son necesarias tres **variables de estado** (dado que $2^3 = 8$). Se necesitan, por tanto, tres biestables para guardar las variables de estado. Para simplificar el diseño, vamos a considerar que las variables de estado, y_2, y_1 e y_0, son al mismo tiempo las salidas, de manera que el estado *A* se representa por $y_2 y_1 y_0 = 000$, el *B* por $y_2 y_1 y_0 = 001$, y así sucesivamente. Para diseñar cualquier sistema secuencial hay que utilizar biestables

(se puede utilizar cualquier tipo de biestable); en el caso de este contador vamos a utilizar biestables tipo *T*, sin que haya razón alguna para elegir este tipo u otro.



(a)

E_n	E_{n+1}
A	B
B	C
C	D
D	E
E	F
F	G
G	H
H	A

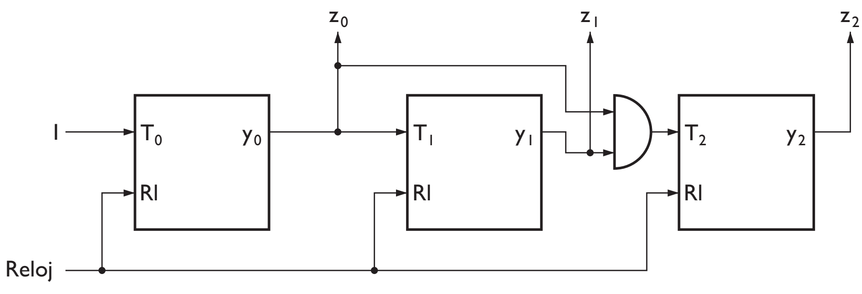
(b)

E	Asignación y_2	y_1	y_0
A	0	0	0
B	0	0	1
C	0	1	0
D	0	1	1
E	1	0	0
F	1	0	1
G	1	1	0
H	1	1	1

(c)

E_n			E_{n+1}					
y_2	y_1	y_0	y_2	y_1	y_0	T_2	T_1	T_0
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1
1	0	1	1	1	0	0	1	1
1	1	0	1	1	1	0	0	1
1	1	1	0	0	0	1	1	1

(d)



(e)

Figura 6.11. *Diseño del contador binario módulo 8: (a) diagrama de estados; (b) tabla de estados; (c) asignación de estados; (d) tablas de transiciones y excitaciones; (e) circuito.*

En la Figura 6.11a se representa el **diagrama de estados** de este sistema secuencial. Este diagrama se construye asignando un nodo (un círculo en nuestro caso) a cada estado y una rama para cada transición entre estados, y representa gráficamente el comportamiento que se desea para este sistema secuencial. La **tabla de estados**, en la que se da el estado

siguiente para cada uno de los estados del sistema y que contiene la misma información que el diagrama de estados, es la de la Figura 6.11b. Teniendo en cuenta las asignaciones de las variables de estado ya indicadas, que se dan en la Figura 6.11c, esta tabla de estados se traduce en la **tabla de transiciones** de la Figura 6.11d (cada estado de la tabla de la Figura 6.11b se ha sustituido por su asignación).

Consideremos una fila cualquiera de la tabla de la Figura 6.11d, la primera por ejemplo. Esta fila nos indica que si el contador está en el estado 000 (es decir, sus salidas son 000), cuando llegue un pulso por la entrada de reloj, el sistema tiene que pasar al estado 001. Es decir, y_2 e y_1 no han de cambiar, e y_0 ha de pasar de 0 a 1 (ha de cambiar). Utilizando la tabla de la Figura 6.7d del biestable T se concluye que la entrada T ha de ser 0 si la salida no ha de cambiar, y 1 si ha de cambiar; esto significa que las excitaciones $T_2T_1T_0$ en el estado 000 han de valer 001. Repitiendo este mismo razonamiento para las diferentes filas se obtienen las funciones T_2, T_1 y T_0 dadas en la Figura 6.11d, de donde:

$$T_2 = y_1 y_0; \qquad T_1 = y_0; \qquad T_0 = 1$$

Utilizando este resultado se obtiene el circuito de la Figura 6.11e.

Es inmediato que con este procedimiento se puede diseñar cualquier contador binario ascendente módulo 2^m. Es fácil anticipar que si se utilizan biestables T, para un contador de 16 serán T_2, T_1 y T_0 como en el contador de 8, y para T_3 se tendrá:

$$T_3 = y_2 y_1 y_0$$

y así sucesivamente para otros valores de 2^m.

Para situar al contador en su estado inicial (000) se pueden utilizar las entradas asíncronas *Clear*. En general se pueden utilizar las entradas asíncronas para escribir en el contador, en paralelo, cualquier valor que se desee, de igual forma que se ha descrito en la Sección 6.3.2 para los registros de desplazamiento.

Estos contadores binarios ascendentes están disponibles en forma integrada, en un solo chip; los hay ascendentes, como el que se ha diseñado, y los hay ascendentes/descendentes, que mediante una señal de control permiten al usuario decidir si la transición siguiente es en sentido ascendente o descendente. Por ejemplo, el circuito 74161, de 16 patas, es un contador binario ascendente de 4 bits, que se puede poner, en paralelo y de forma síncrona, a cualquier valor inicial, y que puede ponerse a cero de forma asíncrona; el chip 74163 es idéntico al 74161 salvo que la puesta a cero se hace de forma síncrona. El chip 74191, de 16 patas, es un contador binario ascendente/descendente de 4 bits, que se puede cargar en paralelo a cualquier valor inicial. El chip 744060, de 16 patas, es un contador binario ascendente de 14 bits.

También hay contadores que cuentan en módulo cualquier entero, sea o no potencia de dos; entre estos están los contadores BCD o decimales, muy utilizados. Por ejemplo, el chip 74160, de 16 patas, es un contador decimal ascendente de 4 bits, que se puede poner, en paralelo y de forma síncrona, a cualquier valor inicial, y que puede ponerse a cero de forma asíncrona; el chip 74162 es idéntico al 74160 salvo que la puesta a cero se hace de forma síncrona. El chip 74190, de 16 patas, es un contador decimal ascendente/descendente de 4 bits, que se puede cargar en paralelo con cualquier valor inicial.

6.5 Diseño de sistemas secuenciales

En el diseño del contador binario ascendente módulo 8 visto en el apartado anterior se han utilizado las herramientas básicas que se usan en el diseño de cualquier sistema secuencial.

En este apartado, mediante el desarrollo detallado de un ejemplo, se aborda la síntesis de cualquier sistema secuencial síncrono, con un procedimiento válido no sólo para contadores.

Partiendo de una **especificación** en la que se indica el comportamiento deseado para el sistema y se fijan los elementos a utilizar en el diseño, la primera etapa del procedimiento consiste en obtener un **diagrama de estados** que responda a esa especificación. Para cubrir esta primera etapa no hay método sistemático alguno, de manera que generar un buen diagrama de estados depende únicamente de la habilidad y experiencia del diseñador.

Tras el pleno convencimiento de que el diagrama de estados responde a las especificaciones, se construye la **tabla de estados** (segunda etapa). El diagrama de estados contiene la misma información que la tabla de estados, pero conviene empezar el diseño obteniendo el diagrama de estados, dada la especial facilidad de la mente humana para captar la información gráfica; sin embargo, para proseguir con el diseño se ha de pasar a la tabla de estados.

Es posible que dos o más estados de la tabla obtenida sean **equivalentes**, en el sentido de que sean indistinguibles por su comportamiento observando únicamente las salidas del sistema. Si así ocurre, se puede reducir el número de estados, pues todos los que sean equivalentes entre sí pueden quedar representados por uno solo. Por tanto, una vez construida la tabla de estados, la siguiente etapa es la **minimización** del número de estados (tercera etapa). El interés en minimizar reside en que lo esperable es que una tabla de estados más simple se traduzca en un circuito más simple y, por tanto, más barato. Hay procedimientos sistemáticos para cubrir esta tarea, procedimientos que no vamos a considerar en esta breve incursión en los sistemas digitales. Como resultado de la minimización se produce otra tabla de estados, usualmente con menos estados que la original.

A continuación se trata de encontrar el circuito que materializa la tabla de estados. Si en ésta hay m estados, siendo:

$$2^{n-1} < m \leq 2^n$$

es claro que hacen falta n variables de estado para codificarlos. La cuarta etapa, conocida como **asignación de estados**, consiste en asignarle a cada estado una combinación específica de las variables de estado. Diferentes asignaciones pueden llevar a circuitos finales diferentes, particularmente con costes diferentes. Por consiguiente no es indiferente cómo se haga la asignación. Hay procedimientos orientativos de cómo hacer la asignación de estados, que no siempre generan indicaciones suficientes como para decidir unívocamente la asignación de estados; en cualquier caso estos procedimientos tampoco se van a considerar en este nivel inicial.

Una vez asignados los estados y decidido el tipo de biestables a utilizar, lo siguiente (quinta etapa) es obtener las **excitaciones de los biestables** y generar las salidas. Con esto se tienen todos los datos necesarios para sintetizar el sistema secuencial síncrono, que se puede materializar con diferentes tipos de circuitos, dependiendo de la disponibilidad y de lo que en cada ocasión interese.

EJEMPLO 6.1

Como ejemplo clarificador de este procedimiento vamos a seguir los pasos anteriores diseñando un circuito sencillo cuya especificación es como sigue.

Especificación: *sea una línea x por la que se transmite información serie sincronizada con la señal de reloj del sistema. Se pide diseñar un circuito secuencial síncrono que detecte la secuencia o patrón 0011.*

EJEMPLO 6.1 (Cont.)

Este circuito tendrá una entrada, x, y una salida, z. Cuando se detecte la secuencia se hará z = 1 (z = 0 en otro caso). Para construir este circuito se dispone de biestables JK y de puertas Y, O y NO.

Diagrama de estados: *en la Figura 6.12a se tiene el diagrama de estados que hemos obtenido para la anterior especificación. Antes de explicar cómo hemos llegado a este diagrama de estados, conviene señalar que en cada rama del diagrama se indica la entrada, e, que provoca cada transición y la salida, s, que se genera en la misma, en la forma e/s.*

Pasando a la creación del diagrama, se ha empezado en un estado inicial A, en el que el sistema se encuentra a la espera del comienzo de la secuencia. Estando en A, en el siguiente pulso de reloj, por x llegará un 0 ó un 1 (ésta es la gran ventaja de los sistemas digitales: sólo hay dos opciones). Si llega un 1, el sistema debe seguir en el estado A, puesto que aún no se ha detectado el primer bit de la secuencia, y la salida ha de ser 0; en otras palabras, del estado A con entrada 1 se pasa al estado A con salida 0, y así se indica en el diagrama de estados con una rama que va de A a A marcada con 1/0. Si en A llega un 0, éste puede ser el primer bit de la secuencia, en cuyo caso se pasa a un nuevo estado, el B, con salida 0 (marca 0/0); el estado B, por tanto, "recordará" que ya se ha detectado el primer bit de la secuencia.

Ya en el estado B, si llega un 0, se puede tratar del segundo bit de la secuencia y pasará a un nuevo estado, el C, con salida 0; C memoriza que han llegado los dos primeros ceros de la posible secuencia. Si en B llega un 1, deberá pasar al estado inicial, A, con salida 0, pues se ha roto la secuencia que estaba empezando a detectarse y de nuevo ha de tratar de detectar dos ceros seguidos.

En el estado C, si llega un 0, deberá permanecer en C, con salida 0, pues los dos últimos bits han sido 0 y podrían corresponder a una secuencia correcta. Si llega un 1, éste será el tercer bit de la secuencia y deberá pasar al nuevo estado D, con salida 0; D recuerda que han llegado 001.

Desde el estado D, si llega un 1, se pasará al estado inicial A tras haber detectado la secuencia (la salida, por tanto, será 1). Si llega un 0, se ha abortado la secuencia en curso de detección. Como el 0 que acaba de llegar puede ser el comienzo de una nueva secuencia, se pasará al estado B, con salida 0.

Y con estos cuatro estados se tiene completo el diagrama de estados, que efectivamente corresponde a la especificación.

Tabla de estados: *en la Figura 6.12b se tiene la tabla de estados correspondiente al diagrama de estados de la Figura 6.12a. Esta tabla de estados consta de dos subtablas, una correspondiente a los estados siguientes para cada combinación de estado presente y de entrada, y la otra en la que se dan las salidas, también para cada estado presente y cada posible entrada.*

A continuación habría que minimizar esta tabla de estados, etapa que no va a cubrirse, dado que en este nivel introductorio no se aborda el problema de la minimización del número de estados. No obstante, en esta ocasión ocurre que la tabla de estados obtenida es ya mínima. Por tanto lo que procede a continuación es tratar de sintetizarla.

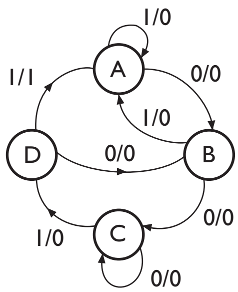
Asignación de estados y tabla de excitaciones: *la asignación que se ha utilizado es A = 00, B = 01, C = 11, D = 10. Con esta asignación resulta la tabla de transiciones y las excitaciones de los biestables que se dan en la Figura 6.12c (recuérdese que se han de utilizar biestables JK). En esta misma tabla se da la función correspondiente a la salida z.*

Las funciones que resultan, tras sintetizarlas con los procedimientos vistos en el capítulo anterior, son:

$J_1 = y_0\overline{x};\ K_1 = \overline{y}_0;\ J_0 = \overline{x};\ K_0 = x;\ z = y_1\overline{y}_0 x$

Y diseñando el circuito correspondiente resulta el de la Figura 6.12d.

En este diseño no se ha prestado atención al posible estado inicial, de manera que cuando se conecta el circuito, los biestables pueden empezar en cualquier valor. Si fuese importante que el sistema empezara en el estado A, bastaría utilizar las entradas asíncronas de los biestables y situarlos inicialmente, antes de que llegue el primer bit por x, al valor 00. Una vez en funcionamiento, cada vez que llega un pulso de reloj los biestables cambian de estado de acuerdo con el valor de la entrada x y con el estado en el que se encuentren, siguiendo el diagrama de estados de la Figura 6.12a.



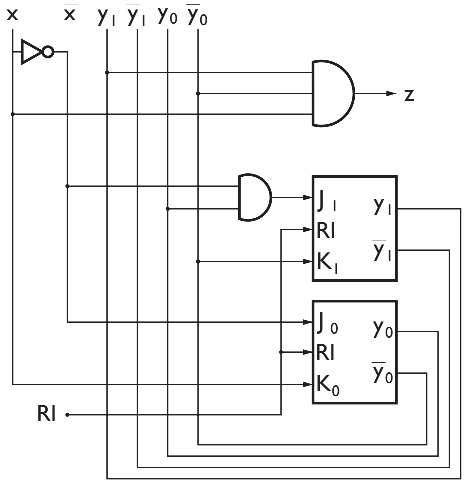
(a)

	x		x	
	0	1	0	1
A	B	A	0	0
B	C	A	0	0
C	C	D	0	0
D	B	A	0	1

(b)

	$y_1^{n+1}\,y_0^{n+1}$		J_1		K_1		J_0		K_0		z	
$y_1^n\,y_0^n$	x = 0	x = 1	0	1	0	1	0	1	0	1	0	1
A → 00	01	00	0	0	–	–	1	0	–	–	0	0
B → 01	11	00	1	0	–	–	–	–	0	1	0	0
C → 11	11	10	–	–	0	0	–	–	0	1	0	0
D → 10	01	00	–	–	1	1	1	0	–	–	0	1

(c)



(d)

Figura 6.12. *Circuito secuencial del Ejemplo 6.1: (a) diagrama de estados; (b) tabla de estados; (c) excitaciones de los biestables y salida; (d) circuito.*

6.6 Memorias RAM

Una memoria **RAM** (*Random Access Memory:* memoria de acceso aleatorio) con 2^n palabras de m bits tiene m entradas de datos $x_0,...,x_{m-1}$, n entradas de dirección, $y_0,...,y_{n-1}$, m salidas de datos, $z_0,...,z_{m-1}$, y $m \cdot 2^n$ celdas de almacenamiento con posibilidad de lectura y escritura, que pueden ser elementos de memoria como el de la Figura 6.1a. Además de las entradas y salidas anteriores, un chip de memoria RAM suele incluir al menos dos señales de control, S (o CS) y R (o WE, o RW), que permiten decidir si se selecciona (S = 1) o no se selecciona (S = 0) la pastilla para una operación de lectura o escritura, y si la operación a realizar es de lectura (R = 1) o de escritura (R = 0). La denominación **acceso aleatorio** quiere decir que en un instante dado se puede acceder a cualquier posición de memoria. En la Figura 6.13a se representa una memoria RAM. Nótese que los bancos de registros descritos anteriormente (Sección 6.3.1) se pueden considerar como memorias RAM con la posibilidad de lectura y escritura simultaneas.

Los elementos de memoria incluidos en una RAM están estructurados en 2^n palabras de m bits. A cada palabra le corresponde una dirección única, de manera que con los n bits de

dirección se decide qué palabra es la que se va a leer o en la que se va a escribir a continuación. Si la operación es de escritura (para ello deben ser S = 1 y R = 0), en la palabra seleccionada se escribe la información que se aplique a las entradas $x_0,...,x_{m-1}$. Si la operación es de lectura (para ello deben ser S = 1 y R = 1), en las salidas $z_0,...,z_{m-1}$, se tiene la información que en ese momento haya en la palabra seleccionada; esta operación de lectura no destruye la información almacenada, que puede volver a leerse cuantas veces se desee.

Las memorias son los chips en los que se consigue mayor densidad de integración (número de transistores por unidad de superficie), y cada vez se están construyendo memorias más rápidas y con mayor capacidad de almacenamiento; de hecho el número de bits que en un chip se pueden memorizar se viene duplicando cada año y medio, aproximadamente, desde que existen las memorias integradas. Un problema ligado a este crecimiento está en el número de terminales que es necesario prever para entradas y salidas. Se utilizan algunos "trucos" para disminuir el número de terminales. Concretamente, complicando un poco la lógica combinacional de control de entrada/salida, los mismos terminales físicos pueden servir tanto de entrada como de salida, decidiendo con las señales de control que, en un determinado instante, actúen como entradas o como salidas. Otra solución utilizada para disminuir el número de terminales de entrada/salida es la **multiplexación temporal de la dirección**, que consiste en dar en dos instantes sucesivos la dirección de la posición de memoria a la que se quiere acceder, primero los bits menos significativos y después los más significativos, por ejemplo, siempre a través de los mismos terminales físicos, almacenándose en un registro interno todos los bits de la dirección; de esta forma se reducen a la mitad los terminales dedicados a dirección, pero se paga el precio de hacer más lento el acceso a memoria; evidentemente esta solución se puede generalizar utilizando más de dos pulsos de reloj para introducir todos los bits de dirección.

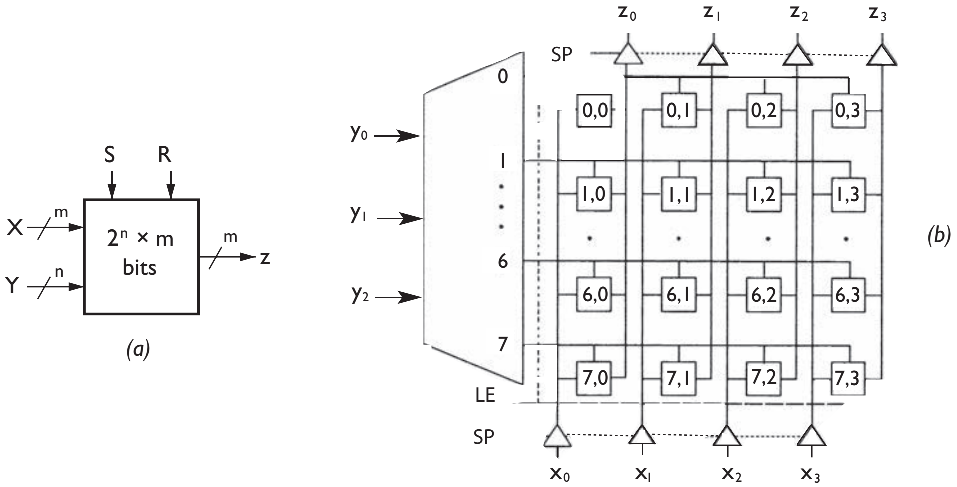


Figura 6.13. *(a) Memoria RAM de 2^n palabras de m bits; (b) organización de una memoria RAM de 8 palabras de 4 bits.*

Por lo que respecta a la estructura interna, una RAM consta básicamente de celdas de almacenamiento, de un decodificador, y de adaptadores para controlar las entradas y salidas, tal como se representa en la Figura 6.13b para el caso de una RAM de 8 palabras de 4 bits

cada una; en este caso hay 3 bits de dirección, 4 entradas de datos, 4 salidas de datos, 32 celdas de almacenamiento, y un decodificador de 3 a 8; con la entrada de control *SP* se selecciona la pastilla, y con *LE* se decide si la operación es de lectura o escritura.

Cuando la pastilla de memoria incluye un número considerable de palabras, esta estructura de la Figura 6.13b con un sólo decodificador para los bits de dirección (lo que podríamos denominar decodificación unidimensional) tiende a producir circuitos en los que predomina enormemente una de las dos dimensiones, como se puede observar en el caso sencillo de la memoria de 16 palabras de 1 bit de la Figura 6.14a, lo que es un inconveniente para su implementación en silicio. Como solución para conseguir circuitos en los que las dos dimensiones del área de silicio sean lo más próximas posibles se utiliza una decodificación bidimensional, con decodificadores de fila y de columna. Esta segunda decodificación de columna se puede realizar con un decodificador propiamente dicho o también con un multiplexor de palabra para obtener la salida y con un demultiplexor de palabra para controlar la entrada, tal como se representa en la Figura 6.14b para este mismo ejemplo de la memoria de 16 palabras de 1 bit; en este caso los 16 bits se han dispuesto en una matriz de 4 filas y 4 columnas; con dos de los bits de dirección, actuando sobre el decodificador, se selecciona una fila de la matriz, y con los otros dos bits de dirección, actuando sobre el multiplexor y el demultiplexor, se selecciona una de las celdas de la fila. No obstante, por lo que respecta a su utilización, es decir, desde el punto de vista externo, de usuario, no hay diferencia alguna entre una RAM con un solo decodificador y otra con dos decodificadores.

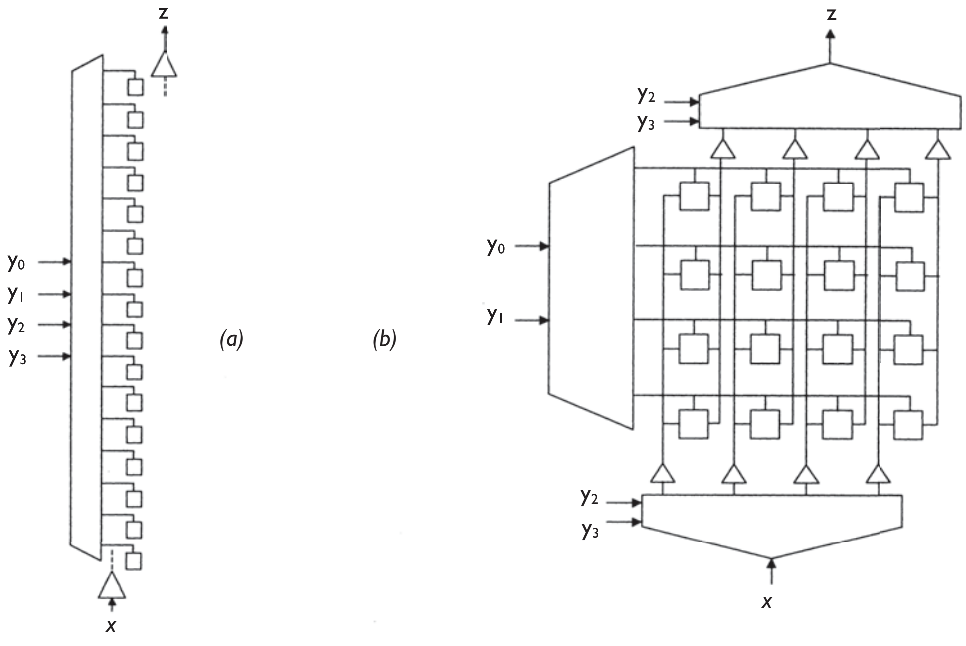
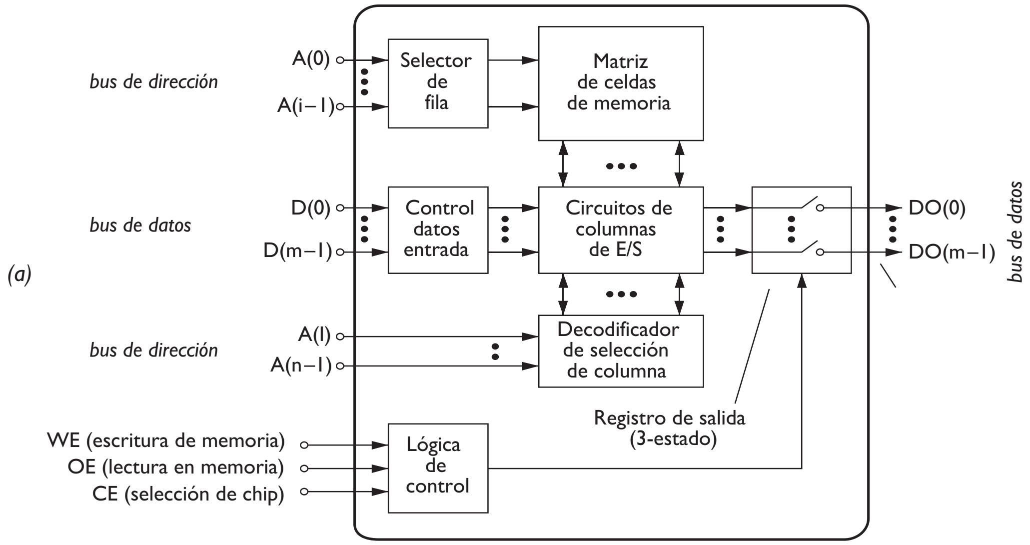


Figura 6.14. *RAM de 16 palabras de 1 bit: (a) con un decodificador; (b) con dos decodificadores.*

Para caracterizar cada chip de memoria se utilizan diferentes parámetros. El primero, obviamente, es su **capacidad**, medida como el número de bits que puede memorizar. Otro parámetro importante es el **tiempo máximo de acceso**, t_{amax}, que es el tiempo máximo que se

emplea en leer o en escribir una palabra. Relacionado con el anterior está el **tiempo de ciclo**, t_C, que es el tiempo mínimo que debe transcurrir entre dos lecturas (se tiene el **tiempo de ciclo para lectura**, t_{CR}) o entre dos escrituras (se tiene el **tiempo de ciclo para escritura**, t_{CW}) sucesivas. Otra medida de la velocidad es el **ancho de banda**, *AB*, que es el máximo número de palabras que se pueden transferir, por segundo, entre la memoria y la CPU; evidentemente, $AB = 1/t_C$. Otro parámetro importante es el **consumo de potencia** de cada chip.

Dependiendo de cómo se construyan las celdas de almacenamiento de las memorias RAM hay dos tipos diferentes, las estáticas y las dinámicas.



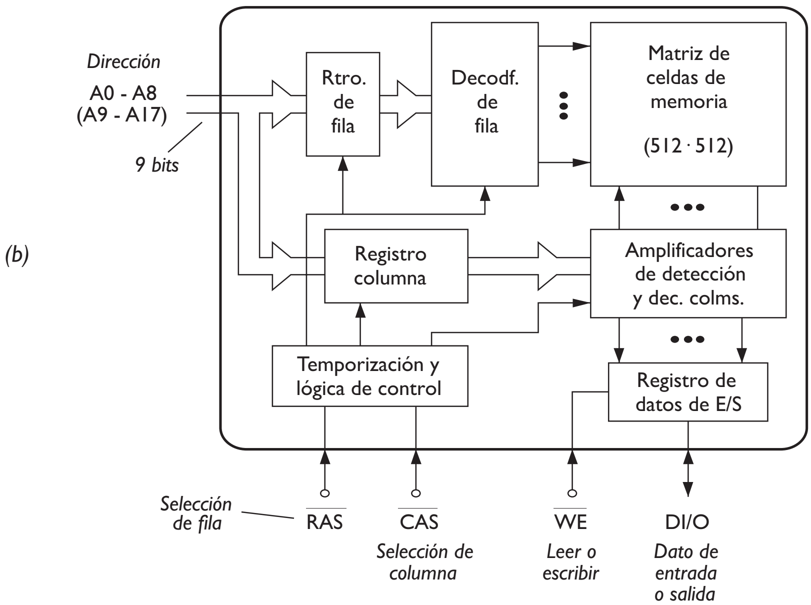


Figura 6.15. *(a) Estructura interna de una SRAM; (b) estructura interna de una DRAM.*

En una RAM estática (**SRAM**) las celdas son básicamente como la celda elemental de la Figura 6.1a; por tanto almacenar un bit consiste en poner un biestable a un determinado valor, al que permanecerá indefinidamente mientras el chip esté funcionando adecuadamente. Cada una de estas celdas SRAM, dependiendo de la tecnología de cada fabricante, consta de 4 a 6 transistores. En la Figura 6.15a se tiene, de forma esquemática, la posible estructura interna de una SRAM.

En una RAM dinámica (**DRAM**) cada celda está formada básicamente por un diminuto condensador más un transistor para controlar la carga o descarga del condensador. Almacenar un bit consiste en cargar (valor 1) o descargar (valor 0) el condensador. Los chips DRAMs, al incluir en cada celda menos transistores que las SRAMs, admiten una mayor capacidad de almacenamiento. Por el contrario, el acceso a una DRAM es más lento que a una SRAM, dado que cargar o descargar un condensador es más lento que cambiar el estado de un biestable. Otra diferencia notable es la siguiente: los condensadores siempre tienen corrientes de fuga, que hacen que se vayan descargando paulatinamente, de manera que una DRAM, para mantener intacta la información escrita en ella, ha de **refrescarla** periódicamente, es decir, cada cierto tiempo ha de recargar aquellas celdas que deban permanecer cargadas. Las SRAMs no necesitan refrescar su contenido. En muchos chips esta operación de refresco se hace automáticamente, de manera que el usuario no ha de preocuparse de la misma, pero siempre consume tiempo, por lo que, de nuevo, siempre las DRAM son más lentas que las SRAM. En la Figura 6.15b se tiene, de forma esquemática, la posible estructura interna de una DRAM.

Si se dispone de chips con RAMs de una determinada capacidad (número de palabras y número de bits por palabra), es fácil interconectarlas para ampliar el número de palabras o el número de bits por palabra. Por ejemplo, es frecuente que los circuitos RAMs comercialmente disponibles tengan palabras de 1 bit; si se quiere construir una memoria con palabras de m bits, es inmediato que para ello basta utilizar m circuitos que almacenen cada uno de los bits de la palabra, todos ellos con los mismos bits de dirección, tal como se representa en la Figura 6.16a. En general, si se dispone de pastillas con 2^n palabras de m bits y se quiere construir una unidad de memoria con 2^n palabras de M bits, siendo $M = k \cdot m$, basta utilizar k pastillas con 2^n palabras de m bits, todas las pastillas con los mismos bits de dirección, de manera que la primera pastilla corresponde a los m primeros bits, la segunda a los m segundos, etc., con la misma estructura que en la Figura 6.16a.

La ampliación del número de palabras consiste en construir un bloque de memoria con 2^{n+N} palabras de m bits utilizando bloques de memoria con 2^n palabras de m bits. Es decir, se trata de pasar de 2^n palabras a $2^{n+N} = 2^n \cdot 2^N$ palabras, o, dicho de otra forma, multiplicar por 2^N el número de palabras original. En esta ampliación el número de bits de dirección pasará de n a $n + N$. Para conseguir la ampliación basta utilizar 2^N chips (o asociaciones de pastillas si también fuese necesario ampliar el número de bits por palabra) de 2^n palabras cada uno, de manera que, utilizando la entrada S de selección de chip de cada uno de los 2^N chips y con un decodificador de N a 2^N, tal como se representa en la Figura 6.16b, en el primer chip se almacenan las 2^n primeras palabras, en el segundo las 2^n segundas, etc. Los n primeros bits de dirección son comunes a todos los chips, mientras que los N últimos son las entradas del decodificador, y en realidad se utilizan para escoger uno de entre los 2^N chips. Con este uso que se hace de la entrada de selección parece perderse la posibilidad de seleccionar individualmente cada chip. Este aparente problema se soluciona inmediatamente

poniendo, en cada entrada de selección de chip, una puerta Y con dos entradas: una será la anterior salida del decodificador y la otra la señal externa de selección de chip.

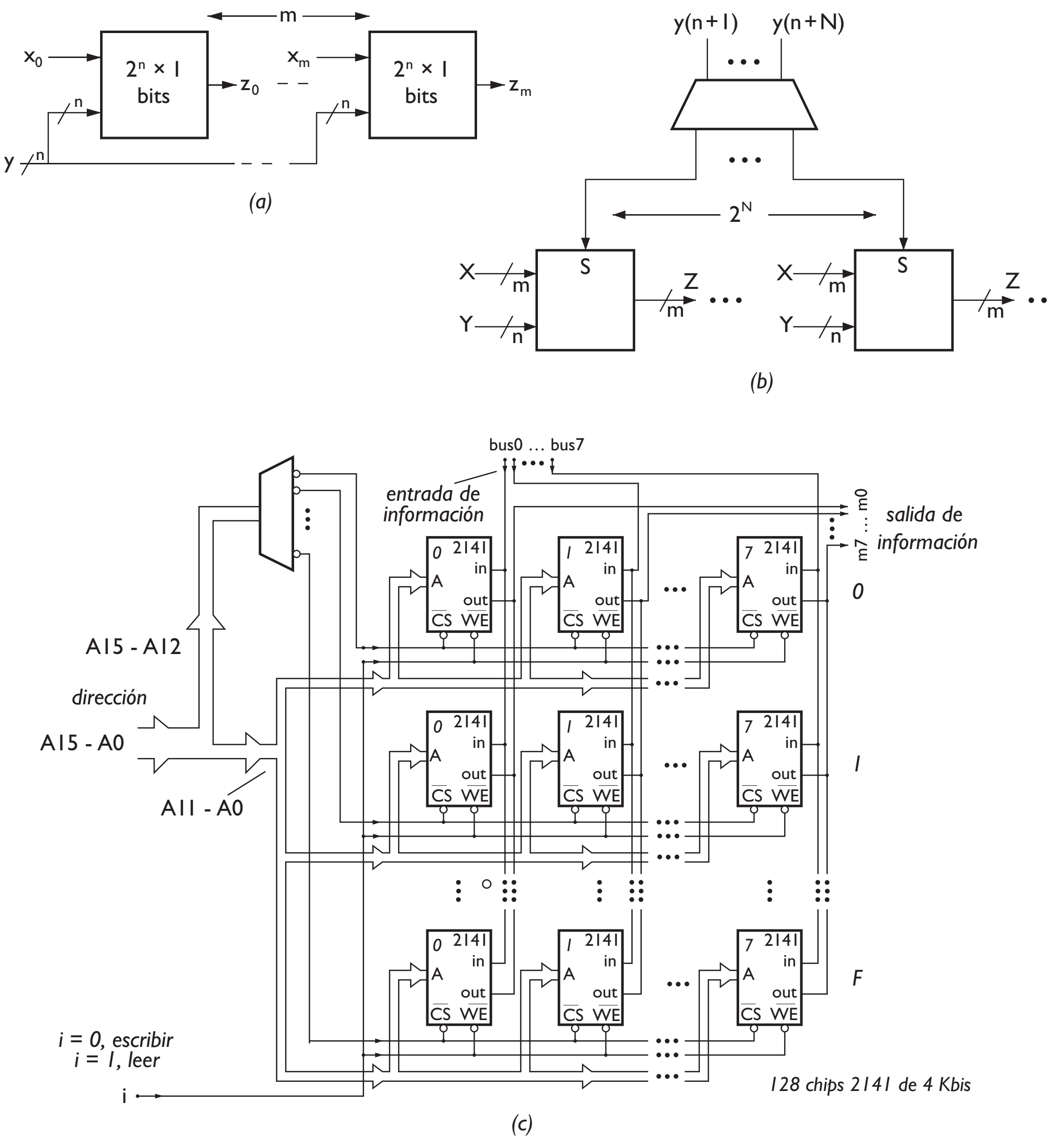


Figura 6.16. *(a) Asociación de memorias RAM para ampliar la longitud de palabra; (b) ampliación del número de palabras; (c) circuito correspondiente al Ejemplo 6.2.*

Esta metodología para ampliar tanto el tamaño de las palabras como el número de palabras que acaba de presentarse para memorias RAM se puede aplicar de la misma forma a memorias ROM.

EJEMPLO 6.2

El chip 2141 es una memoria RAM de 4 Kpalabras de 1 bit cada una. Utilizando estos chips, construir una memoria de 64 K palabras de 8 bits cada una.

Solución: *en este caso hay que pasar de palabras de 1 bit a palabras de 8 bits. Por tanto, cada bloque de 4 Kpalabras de 8 bits estará formado por 8 chips 2141. Además, hay que ampliar el número de palabras, de 4 K a 64 K; es decir, se ha de pasar de 12 bits de dirección (4 Kpalabras) a 16 bits de dirección (64 Kpalabras). Por tanto, como hay que añadir 4 bits de dirección, habrá* $2^4 = 16$ *bloques de 4 Kpalabras de 8 bits. Uniendo estas dos formas de ampliación resulta el circuito de la Figura 6.16c, que incluye un decodificador de 4 a 16 y* $16 \cdot 8 = 128$ *chips 2141.*

6.7 Memorias de acceso secuencial

En contraste con las memorias de acceso aleatorio están las de **acceso secuencial**, en las que, en un determinado instante, sólo se puede acceder a una determinada posición de memoria. De las memorias de acceso secuencial vamos a describir a continuación, por lo interesantes y útiles que resultan, dos tipos concretos: las memorias LIFO y las FIFO.

La lectura de un dato, tanto en una memoria LIFO como en una FIFO, sólo se puede hacer una vez: leída una palabra es como si hubiese desaparecido de la memoria (recordemos que cualquier dato escrito en una RAM puede leerse múltiples veces). Una operación de escritura incrementa en uno el número de datos en la memoria, y una operación de lectura lo decrementa. Evidentemente, la primera operación que debe realizarse siempre en cualquier memoria, tanto de acceso aleatorio como de acceso secuencial, ha de ser una escritura.

6.7.1 Memorias LIFO

En una memoria **LIFO** (***L****ast* ***I****nput-****F****irst* ***O****utput*: última entrada-primera salida) organizada en palabras de n bits hay n entradas de datos, p entradas de control y n salidas de datos. Habrá al menos una entrada de control para indicar que la operación a realizar es de lectura o de escritura, y posiblemente otras entradas de control para seleccionar la pastilla. Funciona de manera que la palabra que se lee en un determinado instante es la última que se escribió y que aun no ha sido leída; es como si los datos se fuesen apilando al escribirlos y, a la hora de extraer o leer uno, se escogiese de la cabecera de la pila; por esta razón las memorias LIFO se conocen también como **memorias pila**.

Utilizando una memoria RAM con un contador binario ascendente-descendente para generar la dirección de la palabra a la que se va a acceder tanto para lectura como para escritura, se puede construir una memoria LIFO, tal como se representa en la Figura 6.17a. Inicialmente el contador se ha de cargar con la dirección de la RAM en la que se vaya a escribir la primera palabra de la LIFO. Las diferentes palabras de la memoria LIFO ocuparán posiciones consecutivas en la RAM, en sentido creciente por ejemplo. Una operación de escritura en la LIFO se implementará escribiendo en la posición de la RAM a la que apunte el contador, seguida de un incremento del contador. Una operación de lectura de la LIFO se implementará decrementando el contador y leyendo la posición de la RAM a la que apunte

el contador. Es decir, en el contador se tiene la dirección de la primera posición de memoria disponible para escritura.

Es inmediato que, cambiando el orden en la implementación de las operaciones de lectura y de escritura, sería igual de eficiente que en el contador se tuviera la dirección de la última palabra escrita, o que las diferentes palabras de la memoria LIFO ocuparan posiciones consecutivas en la RAM, pero en sentido decreciente.

También, utilizando básicamente registros de desplazamiento, es fácil construir una memoria LIFO, como se puede ver en la referencia [Llo03].

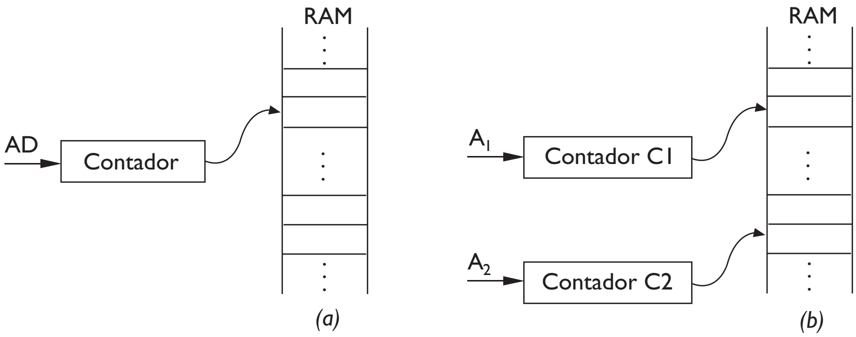


Figura 6.17. *(a) Memoria LIFO; (b) memoria FIFO.*

6.7.2 Memorias FIFO

En una memoria **FIFO** (***F****irst* ***I****nput*-***F****irst* ***O****utput*: primera entrada-primera salida) organizada en palabras de *n* bits hay *n* entradas de datos, *p* entradas de control y *n* salidas de datos. Habrá al menos una entrada de control para indicar que la operación a realizar es de lectura o de escritura, y posiblemente otras entradas de control para seleccionar la pastilla. Funciona de manera que la palabra que se lee en un determinado instante es la primera que se escribió y que aun no ha sido leída; es como si, al escribirlos, los datos se fuesen poniendo en cola y, a la hora de extraer o leer uno, se escogiese de la cabecera de la cola; por esta razón las memorias FIFO se conocen también como **memorias cola**.

Utilizando una memoria RAM con dos contadores binarios ascendentes, C1 y C2, para generar la dirección de la palabra a la que se va a acceder para escritura (C1) o para lectura (C2), se puede construir una memoria FIFO, tal como se representa en la Figura 6.17b. Inicialmente ambos contadores se han de cargar con la dirección de la RAM en la que se vaya a escribir la primera palabra de la FIFO. Las diferentes palabras de la memoria FIFO ocuparán posiciones consecutivas en la RAM, en sentido creciente por ejemplo. Una operación de escritura en la FIFO se implementará escribiendo en la posición de la RAM a la que apunte el contador C1, seguida de un incremento de dicho contador. Una operación de lectura de la FIFO se implementará leyendo la posición de la RAM a la que apunte el contador C2 e incrementando dicho contador.

Si las diferentes palabras de la memoria FIFO ocuparan posiciones consecutivas en la RAM, pero en sentido decreciente, en vez de incrementar los contadores, habría que decrementarlos.

También, utilizando básicamente registros de desplazamiento, es fácil construir una memoria LIFO, como se puede ver en la referencia [Llo03].

6.8 Procedimiento general de diseño

En el diseño de sistemas digitales, tanto combinacionales como secuenciales, se sigue una sistemática que a continuación se va a presentar. En la Figura 6.18 se da un organigrama con el que se resume este procedimiento.

Usualmente cualquier sistema digital surge de una especificación en lenguaje coloquial, con la que se indica, sin una completa precisión, el comportamiento global del sistema. De esta especificación en lenguaje coloquial debe pasarse, si se considera necesario, a la especificación en lenguaje científico, precisando todos los detalles; esta especificación en lenguaje científico se conoce como **definición del problema**, en la que se incluye la decisión con respecto al carácter combinacional o secuencial del circuito a sintetizar. Frecuentemente la definición del problema se plasma en una o varias **tablas verdad**.

De la definición del problema se obtienen las **expresiones analíticas**, que dan las variables de conmutación de salida como funciones de las entradas y, si el sistema es secuencial, de las variables de estado. Hay procedimientos algorítmicos para cada tipo de circuito que facilitan esta traslación de las definiciones a las expresiones analíticas.

Lo normal es que las expresiones analíticas no sean mínimas en el sentido de traducirse en circuitos con el mínimo número de puertas y de elementos de memoria, si es que estos son necesarios. Por esta razón hay que introducir una **etapa de minimización** de las expresiones analíticas, etapa que puede consistir en minimizar las funciones de conmutación utilizando los mapas de Karnaugh, como se ha visto en el capítulo anterior, o bien en la aplicación de otros métodos más sistemáticos, que se verán en otros cursos. De la aplicación de esta etapa resultan las **expresiones mínimas**.

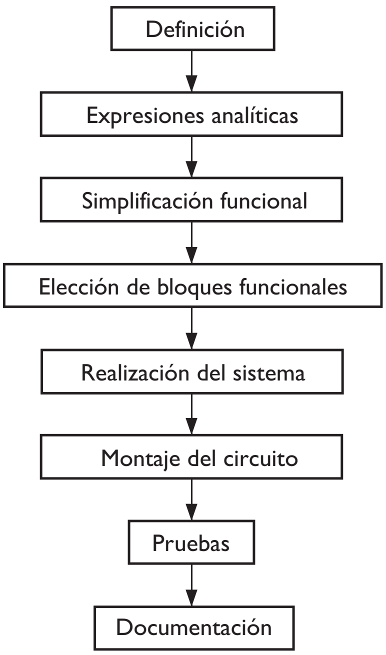


Figura 6.18. *Diagrama correspondiente al procedimiento de diseño.*

Una vez obtenidas expresiones mínimas realizables hay que utilizar un circuito concreto que las materialice. Para ello hay que disponer de unos elementos constructivos con los que ensamblar el circuito final. Es decir, antes de pasar al diseño del circuito deben elegirse los **bloques funcionales** que se van a utilizar. Esta selección de bloques funcionales suele estar decidida por las disponibilidades del laboratorio en el que se esté trabajando. Es frecuente que el tipo de bloques funcionales a utilizar decida el método de minimización que se ha de aplicar para pasar de las expresiones analíticas a las expresiones realizables.

De la conjunción de las expresiones realizables y de los bloques funcionales a utilizar se obtiene la **realización del sistema**, que consiste en dibujar un circuito, formado únicamente con los bloques funcionales, utilizables, y que sintetice las expresiones realizables.

De la realización del sistema se pasa inmediatamente al **montaje del circuito**: se trata de pasar de la realización sobre el papel a la realización física, con circuitos concretos. Es muy aconsejable comprobar previamente el correcto funcionamiento de cada uno de los chips y demás elementos que vayan a utilizarse en el circuito.

Una vez montado el circuito, hay que someterlo a una etapa de **prueba**, con la que se examina su funcionamiento; los ensayos a los que se someta cada circuito dependen del tipo de circuito y de las condiciones ambientales en las que tenga que funcionar. En cualquier caso conviene comprobar al menos que el circuito es funcionalmente correcto, y para ello deben establecerse series de pruebas que permitan decidir al respecto. En el caso más sencillo de circuitos combinacionales con pocas entradas, esta serie de pruebas pueden consistir en aplicar todas las combinaciones de entradas y comprobar que las salidas son correctas. Si el circuito ha de funcionar en condiciones ambientales desfavorables (suciedad, humedad, alta temperatura, etc.) también debe someterse el circuito a las pruebas pertinentes que garanticen que soportará esas condiciones de trabajo. Dentro de este ámbito también deben preverse las pruebas periódicas de funcionamiento a las que deba someterse el circuito, para comprobar durante su vida activa que sigue actuando correctamente; estas pruebas periódicas son tanto más importantes cuanto más crucial sea la acción del circuito. Por ejemplo, si se diseña un sistema digital para la monitorización de una unidad coronaria en un hospital, es muy importante tener la garantía de que el sistema funciona correctamente en todo instante.

Por último, una vez comprobado que el circuito diseñado funciona correctamente, debe generarse una información completa sobre el mismo, en la etapa de **documentación**, pensando que el circuito diseñado ha de ser o puede ser utilizado por otras personas, y que el propio diseñador, con el paso del tiempo, olvida detalles imposibles de reconstruir si no dispone de una documentación correcta. Esta es una etapa muy importante y a la que, indebidamente, se le suele prestar poca atención. Como parte de la documentación estará la descripción de las pruebas periódicas a las que se debe someter al sistema.

Por lo que respecta a los circuitos integrados a utilizar, hay diversas alternativas (que se revisarán a continuación), eligiendo una u otra en función del volumen de producción previsto, de la velocidad requerida para el sistema, de si hay restricciones en el consumo de potencia, de las condiciones ambientales en las que el sistema tenga que funcionar, etc.

La primera alternativa en cuanto a circuitos es utilizar **circuitos estándar** (como los descritos en este capítulo y en el anterior), cuyo diseño lo realiza completamente el fabricante. Dado que se fabrican en grandes series, normalmente tienen un precio muy competitivo y en muchas ocasiones ésta es la alternativa a utilizar. No obstante, si el diseño que se esté realizando requiere prestaciones extra (como bajo consumo, pequeño volumen, alta velocidad), posiblemente haya que recurrir a utilizar circuitos integrados para aplicaciones específicas, o

ASIC (*A*pplication *S*pecific *I*ntegrated *C*ircuits). Entre los circuitos ASIC hay diversos grados de libertad para el usuario, estando incluidos los circuitos programables vistos anteriormente (ROM, PLA, PAL), y otros con posibilidades cada vez más amplias. También cabe pensar en la integración VLSI cuando el volumen de producción vaya a ser muy alto.

6.9 Conclusión

En este capítulo se ha dado una breve visión, general pero al mismo tiempo suficientemente rigurosa, del diseño de sistemas digitales secuenciales. Empezando por la distinción entre sistemas combinacionales y sistemas secuenciales, se estudian a continuación los elementos de memoria, describiéndose con suficiente detalle los diferentes tipos que se usan. Los registros y los contadores se han abordado con adecuada profundidad como para que el lector se sienta cómodo en el manejo de los mismos, aunque no se hayan considerado todas sus posibilidades. Por último las memorias, tan fundamentales en la estructura de los computadores, también se han descrito con suficiente rigor.

Tras el estudio de los dos capítulos dedicados a los sistemas digitales, el lector está capacitado para seguir sin dificultad la descripción de cualquier sistema digital, así como para el diseño de sistemas digitales sencillos.

Por último, como se hace en el capítulo anterior, recomendar la exploración de las páginas web de los diferentes fabricantes de circuitos integrados, que se dan al final del capítulo anterior, y analizar las características de cada uno de los circuitos disponibles en el mercado, tanto los mencionados anteriormente como otros más potentes que están fuera de este primer contacto con los circuitos digitales.

EJERCICIOS

6.1 Sea el circuito de la Figura 6.19. Suponiendo que el retardo de propagación de cada puerta es de 10 ns, dibujar un cronograma de la salida del circuito cuando $x = 1$ durante 100 ns.

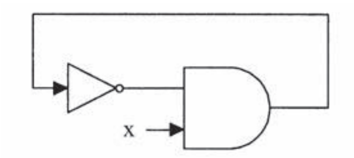


Figura 6.19. *Ejemplo de circuito oscilante.*

6.2 Para un biestable JK disparado por flanco de subida, obtener el cronograma de la salida Q para las señales de entrada representadas en la Figura 6.20.

6.3 Para un biestable D disparado por flanco de bajada, obtener el cronograma de la salida Q para las señales de entrada representadas en la Figura 6.21.

6.4 Para un biestable T disparado por flanco de bajada, obtener el cronograma de la salida Q para las señales de entrada representadas en la Figura 6.22.

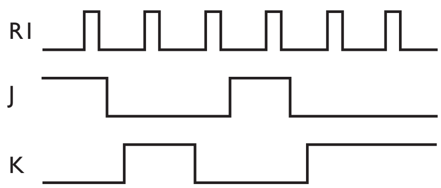


Figura 6.20. *Cronograma para el biestable JK.*

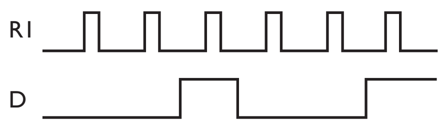


Figura 6.21. *Cronograma para el biestable D.*

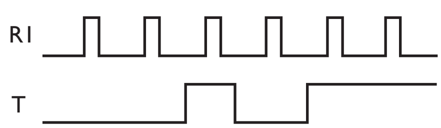


Figura 6.22. *Cronograma para el biestable T.*

6.5 Para un biestable T disparado por flanco de subida, obtener el cronograma de la salida Q para las señales de entrada representadas en la Figura 6.23.

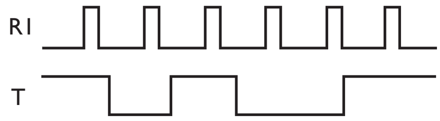


Figura 6.23. *Cronograma para el biestable T.*

6.6 Diseñar un banco de registros de cuatro palabras de dos bits cada una, con dos salidas de datos independientes.

6.7 Utilizando biestables D, diseñar un registro de desplazamiento con entrada serie y salida serie, que permita desplazamientos hacia la derecha o hacia la izquierda (en este caso se debe escribir un cero en el biestable correspondiente a la posición más significativa), según se decida con una señal de control, C.

6.8 Si la salida de un registro de desplazamiento se utiliza como entrada del mismo, se tiene un **contador en anillo**. Dado un registro de desplazamiento de cinco bits funcionando como contador en anillo, obtener todas las secuencias de salida del mismo, dependiendo del estado inicial de los cinco biestables.

6.9 Si el complemento de la salida de un registro de desplazamiento se utiliza como entrada del mismo, se tiene un **contador en anillo con final invertido** (también conocido como **contador Johnson** o como **contador Moëbius**). Dado un registro de desplazamiento de cinco bits funcionando como contador en anillo con final invertido, obtener todas las secuencias de salida del mismo, dependiendo del estado inicial de los cinco biestables.

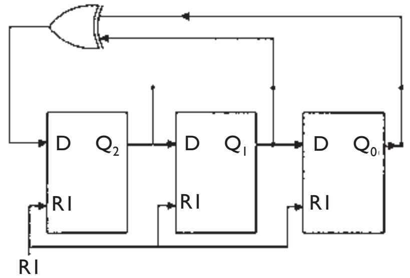


Figura 6.24. *Registro de desplazamiento con realimentación.*

6.10 Dado el registro de desplazamiento de la Figura 6.24 y suponiendo que inicialmente todos los biestables están al valor 1, obtener los diferentes estados por los que pasa este sistema secuencial (es decir, obtener las diferentes configuraciones por las que va pasando el registro de desplazamiento).

6.11 Utilizando biestables tipo T, diseñar un contador binario descendente módulo 8.

6.12 Utilizando biestables JK, diseñar un contador decimal ascendente.

6.13 Utilizando biestables SR, diseñar un contador decimal descendente.

6.14 Utilizando biestables D, diseñar un contador decimal/binario (para cuatro bits) ascendente/descendente. Este contador tendrá dos entradas de control con las que se decidirá en cada instante si cuenta en binario o en decimal, y si la cuenta es ascendente o descendente.

6.15 Diseñar un circuito secuencial con una salida que genere continuamente, como respuesta a los pulsos de reloj, la secuencia de salida 1001000010010000...

6.16 Diseñar un circuito secuencial con tres bits de salida que genere continuamente, como respuesta a los pulsos de reloj, la secuencia de salidas 3, 0, 2, 6, 4, 1, 0 (los tres bits de salida se interpretan como un número binario sin signo).

6.17 Volver a diseñar el circuito del Ejercicio 6.15, pero utilizando un contador en anillo de longitud adecuada.

6.18 Volver a diseñar el circuito del problema 6.16, pero utilizando tres contadores en anillo de longitud adecuada.

6.19 Diseñar un circuito secuencial con una entrada, x, y una salida, z, que cuando detecte la secuencia de entrada 00000000, haga $z = 1$.

6.20 Utilizando básicamente chips de memoria RAM de 1 K palabras de 4 bit, diseñar una memoria RAM de 32 K palabras de 64 bits.

BIBLIOGRAFÍA

[Bae97] Baena, C.; Bellido, M. J.; Molina, A. J.; Parra, M. P.; Valencia, M: *Problemas de circuitos y sistemas digitales*, McGraw-Hill, 1997.

[Bro00] Brown, S.; VRANESIC, Z.: *Fundamentals of Digital Logic with VHDL Design*, McGraw-Hill, 2000.

[Erc99] Ercegovac, M.; Lang, T.; Moreno, J. H.: *Introduction to Digital Systems*, John Wiley & Sons, 1999.

[Flo00] Floyd, T. L.: *Fundamentos de Sistemas Digitales*, Prentice Hall, 1997.

[Gaj97] Gajski, D. D.: *Principios de Diseño Digital*, Prentice Hall, 1997.

[Kat94] Katz, R. H.: *Contemporary Logic Design*, The Benjamin/Cummings Publishing Company, Inc, 1994.

[Lal96] Lala, P. K.: *Practical Digital Logic Design and Testing*, Prentice Hall, 1996.

[Llo03] Lloris, A., Prieto, A., Parrilla, L.: *Sistemas Digitales*, McGraw-Hill, 2003.

[Nel95] Nelson, V. P.; Nagle, H. T.; Carroll, B, D.; Irwin, D.: *Digital Logic Circuit Analysis & Desing*, Prentice Hall, 1995.

[Tin00] Tinder, R. F.: *Engineering Digital Desing*, Academic Press, 2000.

[Vin04] Vingron, S. P.: *Switching Theory, Springer*, 2004.

CAPÍTULO 7

Esquema de funcionamiento de un computador

Una vez que en los capítulos anteriores se ha realizado una introducción al nivel conceptual de la lógica digital, con los elementos y módulos allí definidos podemos pasar a considerar al computador en un nivel de abstracción superior: el nivel de micromáquina. En efecto, en este capítulo se analiza la estructura y el funcionamiento de los computadores suponiendo que se dispone de unidades de memoria, ALU, biestables, registros, contadores, etc. El capítulo hace especial hincapié en los elementos internos de un procesador y en la temporización o ciclo que aquél realiza para ejecutar las instrucciones máquina. También se introduce el concepto de interrupción, analizando las causas de su producción y cómo se gestionan o atienden las peticiones de interrupción.

7.1 Conceptos previos

En el Capítulo 1 (Sección 1.2) se describieron sucintamente las distintas unidades funcionales de un computador, que son las que se muestran esquemáticamente en la Figura 7.1 y en la Tabla 7.1.

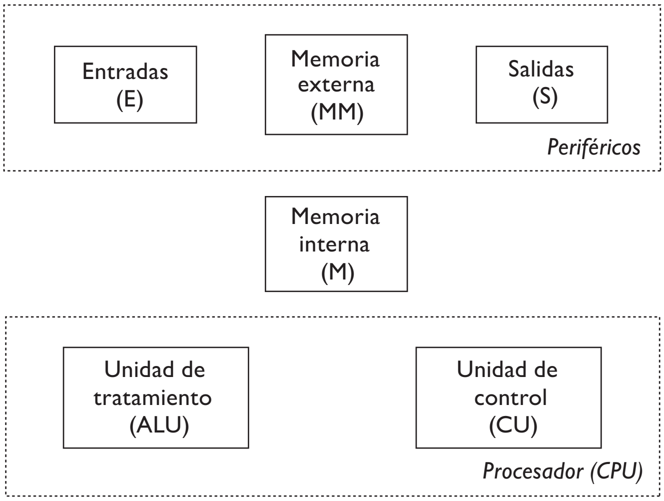


Figura 7.1. *Unidades funcionales de un computador.*

Tabla 7.1. *Esquema de las unidades funcionales de un computador.*

Unidades centrales	– Procesador o unidad central de procesamiento (CPU)	• Unidad de tratamiento o camino de datos (contiene la ALU) • Unidad de Control (CU)
	– Memoria interna (M)	
Periféricos	– Memoria externa (discos magnéticos, ópticos y cintas) (MM)	
	– Dispositivos de entrada (E)	
	– Dispositivos de salida (S)	

Estas unidades se analizarán con más detalle en los Capítulos 9 a 11; no obstante aquí, en este capítulo, se incluyen algunos conceptos básicos para comprender adecuadamente cómo funciona un computador.

Los elementos del procesador y de la memoria principal, considerados en el nivel conceptual de micromáquina, están constituidos por elementos del nivel de lógica digital vistos en los Capítulos 5 y 6 (puertas lógicas, decodificadores, codificadores, multiplexores, demultiplexores, circuitos aritméticos, elementos de memoria, registros, PLAs, etc.). Todos los computadores convencionales contienen algunos de estos elementos con objetivos muy concretos, recibiendo por ello nombres específicos. A continuación vamos a repasar brevemente, y sólo desde un punto de vista funcional, los elementos que se utilizarán en este capítulo.

Biestables

Un biestable es sencillamente un elemento de memoria que almacena un bit de información. Los que se usan en este capítulo son de tipo D (Sección 6.2). El hecho de dar a un biestable, *Z,* por ejemplo, el valor lógico 1, se representa así:

[7.1]

$$Z \leftarrow 1$$

Registro

En las distintas unidades del computador con gran frecuencia es necesario almacenar temporalmente un dato, una instrucción o una dirección individuales de un número determinado, *n,* de bits. Para este cometido se utiliza un circuito que se denomina **registro** (Sección 6.3). La Figura 7.2 representa un registro. Dispone de un conjunto de *n* entradas por donde, a través de un bus de ancho *n,* llega el dato o instrucción a almacenar, de otro conjunto de *n* salidas donde constantemente aparece la información que está grabada en el registro, y una entrada de 1 bit para una señal de control (*c,* en la figura). Simplificadamente, cuando $c=0$ el registro no cambia su contenido, independientemente de lo que haya en sus entradas; sin embargo, en el momento de hacer $c=1$ la información que haya en sus entradas queda almacenada en el interior del registro y ésta aparece inmediatamente en sus salidas. La carga de un registro, por ejemplo *R7,* con una información procedente de otro registro, por ejemplo *RM,* se representa así:

[7.2]

$$R7 \leftarrow RM$$

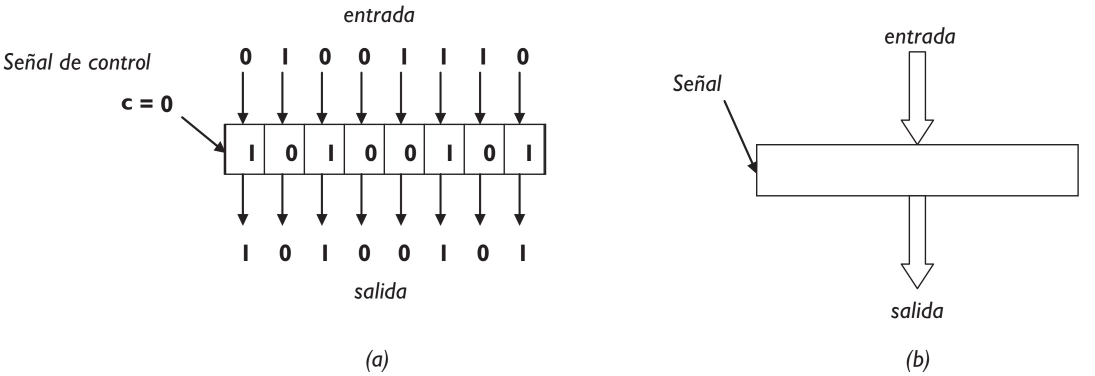


Figura 7.2. *Esquema simplificado de un registro(a), y símbolos (b) y (c) utilizados para su representación.*

Contadores

Los contadores (Sección 6.4) que consideraremos en este capítulo, son contadores binarios ascendentes y con la posibilidad de carga en paralelo. Es decir, cuando reciben una señal de control *t* se incrementa su valor binario en una unidad; en otras palabras, la secuencia que se genera es la siguiente:

00...00; 00...01; 00...10; 00...11; ...; 11...00; 11...01; 11...10; 11...11; 00...00; 00...01; ....

Como veremos más adelante definiremos dos contadores, que se denominan *PC* y *SP*. El incremento del contador PC se describe así:

[7.3]

$$PC \leftarrow PC + 1$$

La carga en paralelo de PC con el contenido (salida), por ejemplo, del registro DR, se denota como:

[7.4]

$$PC \leftarrow DR$$

Unidad de tratamiento (ALU)

La **unidad de tratamiento** (o **camino de datos**) está formada por la **unidad aritmético lógica (ALU)** y otros elementos auxiliares por donde se transmiten o en donde se almacenan temporalmente los datos al objeto de operar con ellos. Así es normal que contenga un **banco de registros**, usualmente de 8 a 256, donde se almacenan los datos y resultados parciales más utilizados dentro de una secuencia de operaciones, pudiéndose reducir así los accesos a la memoria principal, que son siempre más lentos.

Usualmente la ALU (Sección 5.4.10) opera en paralelo con datos, de una determinada longitud, que se proporcionan a través de dos buses de entrada, y proporciona el resultado a través de un bus de salida, como se indica en la Figura 1.3. La ALU dispone de otras entradas a través de las cuales la unidad de control especifica la operación concreta que debe realizar con los datos de entrada en un momento dado. Una operación, suma por ejemplo, en la ALU se representa así:

[7.5]

$$BS \leftarrow BA + BB$$

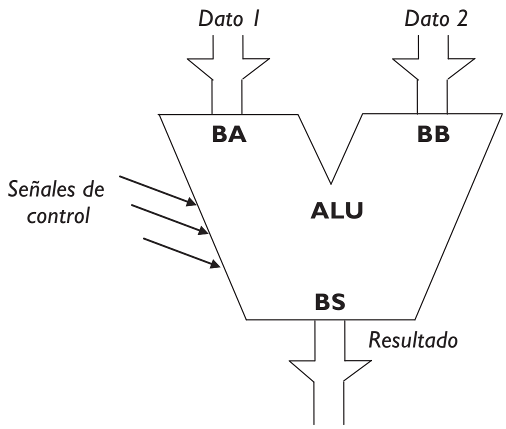


Figura 7.3. *Símbolo para representar una unidad aritmético-lógica (ALU).*

Memoria central

Según indicamos en las Secciones 1.2 y 6.6, la memoria está dividida en **posiciones** (denominadas también **palabras de memoria**) de un determinado número de bits n, que es donde se almacena o memoriza la información.

La entrada y salida de datos en la memoria se puede hacer con un bus bidireccional, que se utiliza para ambas funciones o, tal y como indica la Figura 7.4, con buses específicos de entrada de datos (para el dato a grabar) y de salida de datos (para el dato leído). Estos buses son de n bits de forma que en un instante dado se leen los bits de una posición completa de memoria (en la Sección 10.2.2 se ampliará esta cuestión). Además, la memoria dispone de un **bus de direcciones** de m bits; pudiendo direccionar en total hasta 2^m posiciones distintas de memoria. La capacidad máxima posible de una memoria será:

[7.6]

$$C_{MP} = 2^m \ palabras = \frac{2^m \cdot n}{8} Bytes$$

EJEMPLO 7.1

Supóngase que un computador puede llegar a tener una capacidad máxima de memoria de 512 MBytes y que la longitud de palabra es de 32 bits. Determinar el número de bits que tendría que tener el bus de direcciones.

El número de palabras de la memoria será:

$$C_{MP} = \frac{512\,MB \cdot 8\,bits/Byte}{32\,bits/palabra} = 128\,Mpalabras = 2^7 \cdot 2^{20} \ palabras = 2^{27} \ palabras$$

Es decir, el número de bits del bus de direcciones tendría que ser de 27 bits.

En general, la memoria dispone de dos señales de control. Una de ellas, $R/\overline{W}$, sirve para especificar si se desea leer ($R/\overline{W} = 1$) o escribir (almacenar) ($R/\overline{W} = 0$) en la memoria. También, es habitual que haya otra señal de control para seleccionar la memoria, que en

este y sucesivos capítulos denominaremos $IO/\overline{M}$. Para que la memoria lea o escriba necesariamente ha de ser $IO/\overline{M}=0$. Para leer o escribir una palabra de memoria en una posición determinada, la unidad de control debe generar las señales de control $R/\overline{W}$ e $IO/\overline{M}=0$, y hacer que llegue al bus de direcciones la dirección de la posición a la que se quiere acceder, y al bus de entrada de datos el dato a escribir. En respuesta a estas entradas la memoria realiza la operación correspondiente: si $R/\overline{W}=1$ escribe en la posición indicada el dato de entrada, y, si $R/\overline{W}=0$ proporciona por el bus de salida de datos la palabra almacenada en la posición indicada en la entrada.

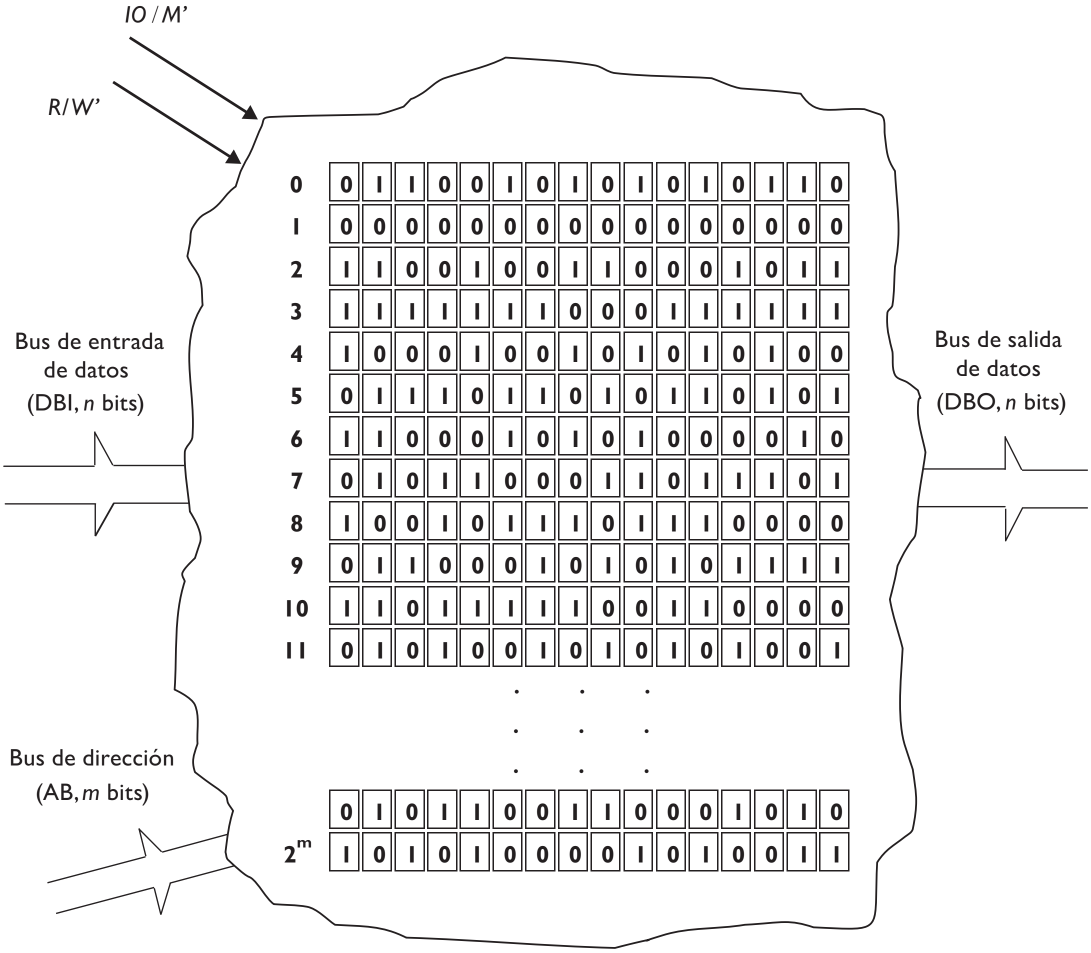


Figura 7.4. *Diagrama simplificado de una memoria principal.*

Supongamos que la dirección de memoria se encuentra en el registro AR y un dato a escribir en dicha posición en DR. La operación de escritura en memoria se denota así:

[7.7]

$$M(AR) \leftarrow DR$$

La operación contraria, es decir, la lectura del dato de la posición AR y su almacenamiento en DR se simboliza de la siguiente manera:

[7.8]

$$DR \leftarrow M(AR)$$

Aunque la memoria principal es muy rápida (tiempos de acceso a una posición de memoria de unas decenas de nanosegundos) es del orden de 50 veces más lenta que la unidad de control y la ALU (unos nanosegundos), por lo que estas últimas unidades se ven frenadas considerablemente cuando tienen que captar o escribir una palabra de memoria. Para paliar este problema se utiliza una memoria de tecnología más rápida intermedia entre la memoria principal y las unidades de control y tratamiento, que es unas 10 veces más rápida que la memoria principal pero, al ser más cara y consumir mayor energía eléctrica, unas 500 veces menos capaz que ésta. Esta memoria se denomina **memoria caché** (Sección 10.2.1). En definitiva, la memoria caché es usada por el sistema operativo para mantener las palabras de memoria más comúnmente usadas por la unidad de control o la ALU, evitando así los relativamente lentos accesos a la memoria principal.

Puertos de E/S

En el nivel de micromáquina cada periférico, ya sea de entrada, salida o de memoria masiva, está constituido por uno o varios puertos de entrada/salida. Un puerto es sencillamente un registro a través del cual el procesador o la memoria central intercambian información con el periférico. Al igual que las posiciones de memoria, cada puerto tiene asociada una dirección; de forma tal que cuando el procesador envía o capta una información de un registro, debe proporcionar la dirección del puerto de que se trata. Además se suele utilizar dos señales de control:

- $IO/\overline{M}$ que especifica si la dirección facilitada se refiere a la memoria principal $(IO/\overline{M}=0)$; o a un puerto de E/S ($IO/\overline{M}=1$), y
- $R/\overline{W}$; que especifica si se el procesador lee de un puerto de entrada ($R/\overline{W}=1$), o escribe en un puerto de salida ($R/\overline{W}=0$).

En este capítulo, por lo tanto, los puertos representan a los periféricos, y se denotarán como IPv el puerto de entrada v, y como OPv el puerto de salida v.

Buses

Según indicamos en el Capítulo 1, la interconexión entre los distintos elementos del computador se realiza mediante hilos conductores en paralelo que transmiten simultáneamente información (una palabra de datos, por ejemplo). De esta forma se consigue una velocidad razonable de funcionamiento. Un conjunto de cables que transmite información en paralelo se denomina ***bus***. En la Figura 7.5 se muestran tres formas distintas de representar un bus de 8 hilos, y que por tanto transmite en paralelo datos de 8 bits.

Los buses que interconectan las distintas unidades funcionales se componen de tres sub-buses: sub-bus de datos, sub-bus de direcciones y sub-bus de control.

El **sub-bus de datos** transporta los datos que se transfieren entre las unidades, y su número de hilos determina al longitud de palabra del computador; es decir, si un computador contiene un sub-bus de datos de 16 hilos para transmitir simultáneamente datos de 16 bits se dice que está organizado en palabras de 16 bits. El sub-bus de datos suele ser bidireccional; esto es, los mismos hilos se utilizan para transmitir información hacia adentro o hacia afuera de una unidad (el procesador, por ejemplo), en instantes diferentes.

El **sub-bus de direcciones** transporta la dirección de la posición de memoria o del puerto de E/S que interviene en el tráfico de información (de donde procede el dato o a donde se dirige).

El **sub-bus de control** contiene líneas conductoras que transportan las señales de control y las señales de estado, indicando la dirección de la transferencia de datos, coordinando la temporización de eventos durante la transferencia, transmitiendo señales de interrupción, etc.

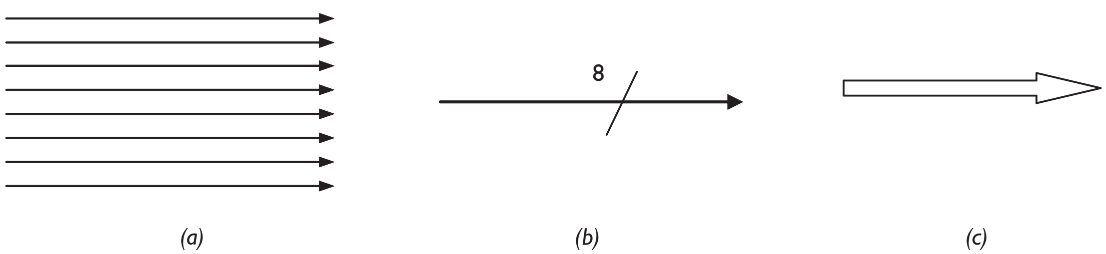


Figura 7.5. *Ejemplo de bus de 8 hilos y símbolos utilizados para su representación.*

En la Figura 7.6 se muestra un esquema que interconecta todas las unidades mediante un único bus. Entre las líneas del sub-bus de control se encuentran las de las señales $R/\overline{W}$ e $IO/\overline{M}$, y cuyo significado se resume en la Tabla 7.2. Además del bus indicado, el procesador y las demás unidades, contienen buses internos que interconectan los elementos que lo constituyen.

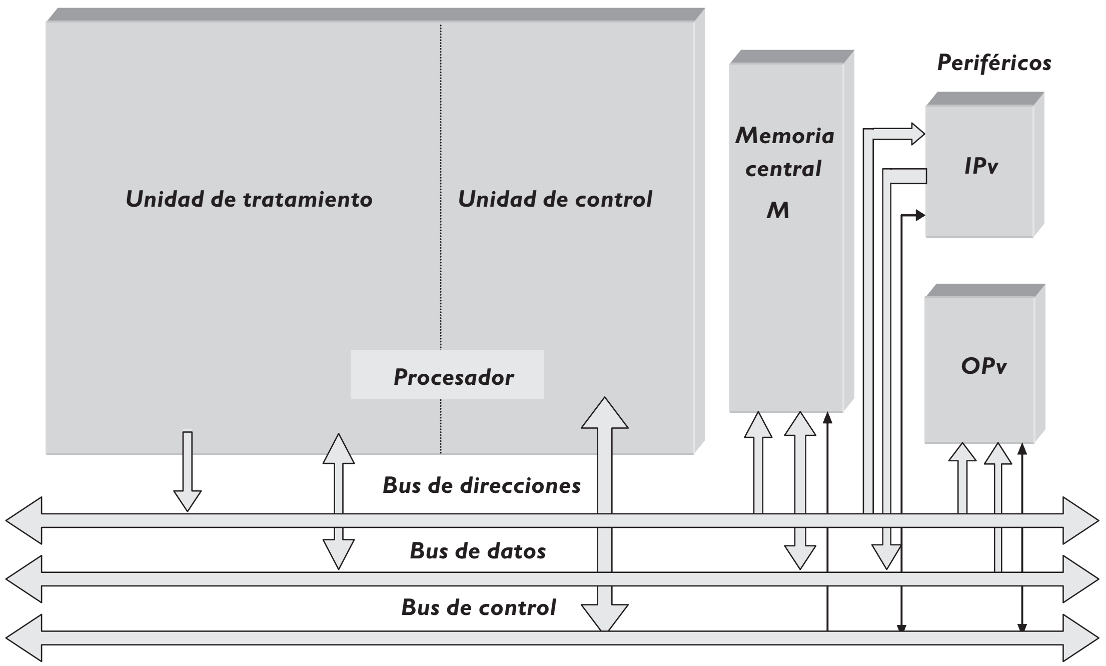


Figura 7.6. *Interconexión de las distintas unidades funcionales de un computador con un bus único.*

Tabla 7.2. *Acciones que controlan las señales* $R/\overline{W}$ e $IO/\overline{M}$.

$R/\overline{W}$	$IO/\overline{M}$	Operación
0	0	Escribir en memoria
0	1	Escribir en un puerto de salida
1	0	Leer de memoria
1	1	Leer de un puerto de salida

7.2 Funcionamiento interno de un computador

En los Capítulos 8 y 9 se trata ampliamente esta cuestión; no obstante, se incluyen en este capítulo de introducción algunas ideas a este respecto, que sirven para comprender mejor los conceptos que se expondrán en las próximas secciones.

Supongamos que tenemos un programa escrito en lenguaje máquina, porque lo hemos hecho directamente así o porque previamente ha sido traducido de un lenguaje de alto nivel por un compilador.

Para ejecutarlo, lo primero que hay que hacer es introducirlo en memoria. De esta función se encarga un programa del sistema operativo denominado cargador. El **cargador** *introduce el programa en posiciones consecutivas de memoria, a partir de una dada*.

Una vez cargado el programa, por ejemplo a partir de la posición **PC** de memoria, se indica al computador que *pase su control a la posición* ***PC*** *de memoria.* Esta operación la inicia el sistema operativo enviando al procesador una instrucción de salto o, en ciertos computadores, también puede hacerse manualmente con interruptores del panel de control. A partir de ese momento la unidad de control repite sucesivamente las dos siguientes fases:

- **a) Fase de captación de la instrucción.** Lleva de la memoria (M) a un registro de la unidad de control (llamado ***IR***) la instrucción que está en la posición *PC.* Cambiar el valor de *PC* por *PC+1.*
- **b) Fase de ejecución de la instrucción.** La Unidad de Control (CU) interpreta el código de operación de la instrucción y, según sea éste y las señales de estado, envía señales de control a las unidades y circuitos que deben intervenir para ejecutar la instrucción, efectuando las operaciones (microoperaciones) que ésta implica. Volver a la fase (a).

Si la ejecución de la instrucción implica saltar a otra instrucción, en la posición *m,* por ejemplo (alterándose por tanto el orden secuencial), la CU, en la fase de ejecución de la instrucción de salto, hace que se cambie el contenido que hubiese en *PC* por el valor *m,* de forma que en la siguiente fase de captación se ejecuta la instrucción que está en *m,* por ser este el valor actual de *PC.* Este sería el caso de una instrucción de las que se incluyen en el grupo de "bifurcación y saltos" (Sección 1.4).

Supongamos, por ejemplo, que se dispone de un computador que tiene un teclado como unidad de entrada, y una pantalla como unidad de salida (Figura 7.7), y que su lenguaje máquina contiene, entre otras, las siguientes instrucciones:

- Entrada de información llevándola desde un dispositivo de entrada, *IP(v),* a un registro*, rx,* de la unidad de procesamiento. Abreviadamente representaremos esta instrucción así: ***IN rx,IPv***. El código de operación de esta instrucción es: ***0100***.

- Salida de información procedente de un registro*, rx,* de la unidad de procesamiento a un dispositivo de salida*, OPv.* Abreviadamente esta instrucción la representaremos como: ***OUT OPv,rx***, y su código de operación es ***0101.***
- Sumar en la ALU el contenido del registro *rs* con el registro *ra* y ubicar el resultado en el registro *rx.* Abreviadamente esta instrucción la representamos como ***ADDS rd,rs,ra***, y su código de operación es ***0110***.
- Finalizando el programa, el computador detiene su funcionamiento, quedando a la espera de realizar otra tarea. Esta instrucción abreviadamente la representamos por ***HALT,*** y su código de operación es ***1111***.

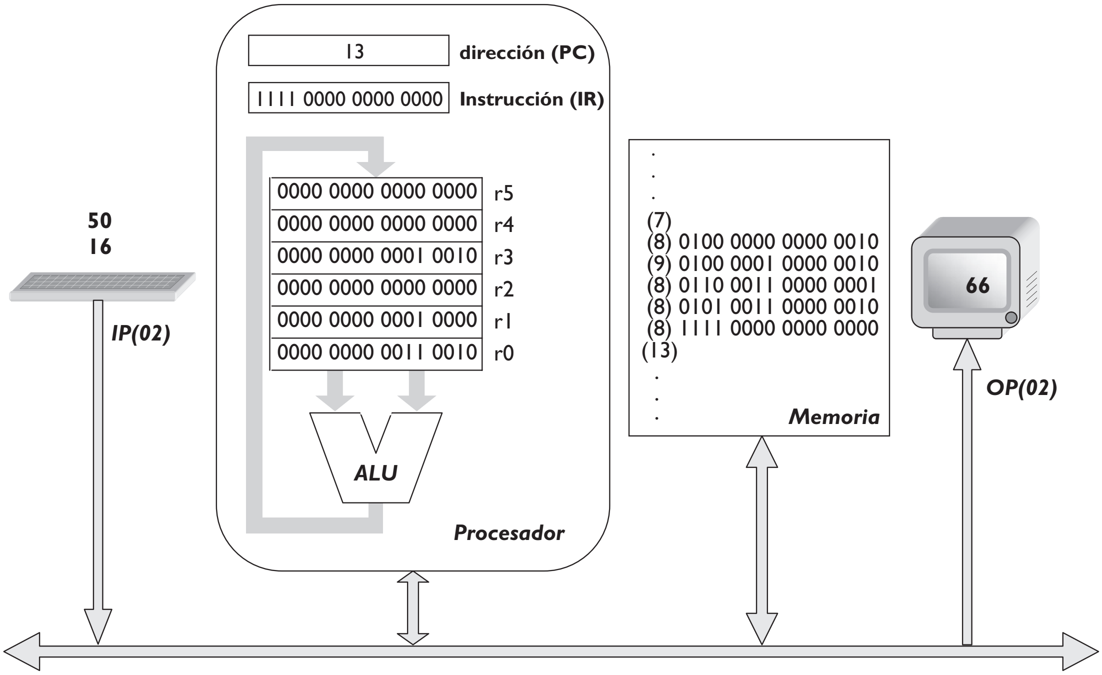


Figura 7.7. *Esquema del computador del ejemplo, y contenidos de la memoria y de los registros después de ejecutado el programa.*

Se desea efectuar un programa para sumar dos números que se proporcionarán a través del teclado de entrada. Analizando el repertorio de instrucciones de que se dispone, para sumar los dos números, primero el computador tiene que leer *(dar entrada a)* los dos números llevándolos a dos registros, luego realizar la suma en la ALU, dejando el resultado en otro registro, y después sacar el resultado a través de una unidad de salida. Vamos a utilizar como dispositivo de entrada el teclado, que en nuestro computador es el dispositivo *IP(02)*; y como dispositivo de salida la pantalla que, en este caso, suponemos que es el dispositivo de salida *OP(02).* Las operaciones se pueden representar con el diagrama de la Figura 7.8.

El programador, en lenguaje máquina, ha de determinar los registros a utilizar. En el ejemplo se pueden seleccionar así:

Primer sumando,	en registro r0
Segundo sumando,	en registro r1
Suma,	en registro r3

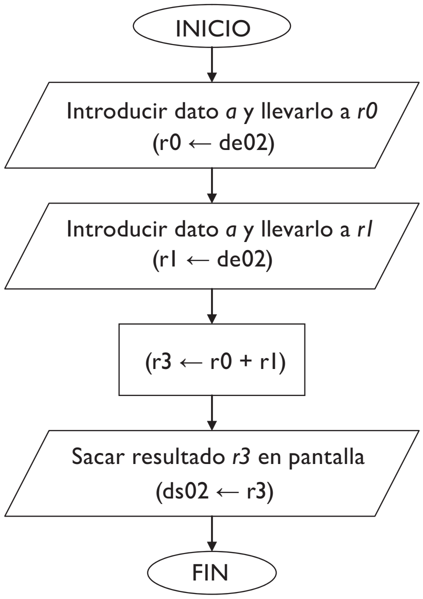


Figura 7.8. *Diagrama de las operaciones en lenguaje máquina a realizar para ejecutar el programa de suma.*

El programa es:

1) Introducir a través del teclado, *IP(02),* el primer dato y llevarlo al registro *r0*.
   IN r0,IP02 instrucción máquina: **0100 0000 0000 0010**

2) Leer el segundo dato y llevarlo al registro *r1*.
   IN r1,IP02 instrucción máquina: **0100 0001 0000 0010**

3) Sumar en la ALU los contenidos de los registros *r0* y *r1*, y el resultado dejarlo en *r3*.
   ADDS r3,r0,r1 instrucción máquina: **0110 0011 0000 0001**

4) Sacar por la pantalla, *OP(02)* el resultado, que está en *r3*.
   OUT OP02,r3 instrucción máquina: **0101 0011 0000 0010**

5) Fin de programa; es decir, detener el funcionamiento de la computadora.
   HALT instrucción máquina: **1111 0000 0000 0000**

Después de describir cada instrucción hemos incluido su significado, en abreviaturas, y la instrucción máquina, que se forma con16 bits (es un computador estructurado en palabras de 16 bits): los cuatro primeros son el código de operación y los doce restantes especifican los lugares en donde se encuentran o donde hay que llevar los datos (registro r0: 0000; registro r1: 0001; registro r3: 0011, y dispositivo 02: 00000010). En definitiva, el programa en lenguaje máquina contiene cinco instrucciones, y es el siguiente:

0100 0000 0000 0010
0100 0001 0000 0010
0110 0011 0000 0001
0101 0011 0000 0010
1111 0000 0000 0000

Supongamos que el programa se carga en memoria a partir de la dirección *PC=8* (tal como se muestra en la Figura 7.7), y se indica al computador que pase su control a la instrucción que está en 8. En la Figura 7.7 se muestra el contenido de la memoria después de ejecutar el programa. Por simplicidad y claridad hemos representado las direcciones en decimal; en la realidad tanto las instrucciones como los datos de entrada y salida son palabras de 16 bits (16 ceros o unos). Los pasos de la ejecución pueden seguirse fácilmente con ayuda de la Figura 7.7 y Figura 7.8; son los siguientes:

(1.a) La CU capta la instrucción que está en *PC=8,* y la lleva a *IR*. Después *PC* se cambia a *PC = 8+1 = 9*.

(1.b) La CU interpreta el código de la instrucción captada, **0100**. Siempre que reciba este código de operación da las señales de control adecuadas para que el dispositivo de entrada (en este caso el *IO(02),* con código **0000 0010**) lea un dato y sea llevado a un registro (al *r0,* de código **0000**). Si el dato tecleado es 50, al final de la ejecución de la instrucción este valor en binario (**0000 0000 0011 0010**) queda grabado en el registro *r0*.

(2.a) La UC capta la instrucción que está en *PC=9*, llevándola a *IR*. Después *PC* cambia a *PC = 9+1 = 10*.

(2.b) La UC interpreta el código de la instrucción captada, **0100**. Es otra instrucción de entrada y genera las mismas señales de control que en la instrucción anterior. En este caso, se capta el segundo dato (16, por ejemplo) y queda grabado en binario (**0000 0000 0001 0000**) en *r1* (código **0001**).

(3.a) La CU capta la instrucción que está en *PC=10,* y la lleva al registro *IR*. Después hace *PC = 10+1 = 11.*

(3.b) La CU interpreta el código de operación de la instrucción **0110**, generando las señales de control necesarias para que se efectúe, en la ALU, la suma de los datos de los registros *r0* (código **0000**) y *r1* (**0001**), llevándose el resultado al registro *r3* (código **0011**). En consecuencia, en *r3* queda almacenado la suma (*0000 0000 0011 0010 + 0000 0000 0001 0000 = 0000 0000 0100 0010;* en decimal *66*).

(4.a) La CU capta la instrucción que está en *PC=11,* y la almacena en *IR*. Después hace *PC = 11+1 = 12*.

(4.b) La CU interpreta el código de operación de la instrucción captada, en este caso **0110**, que corresponde a una instrucción de salida, y genera las señales de control adecuadas para que el contenido del registro especificado en la instrucción, es decir *r3* (**0011**), sea llevado al dispositivo de salida *OP(02)* (código **0000 0010**). En consecuencia, el resultado de la suma *0000 0000 0100 0010* aparecerá en la pantalla, en decimal (66).

(5.a) La CU capta la instrucción que está en *PC=12,* llevándola a *IR*. Cambia *PC* a *PC = 12+1 = 13*.

(5.b) El código de operación, en este caso **1111**, es interpretado por la CU, dando ésta las señales de control adecuadas para que el computador se detenga, dando por concluido el programa.

El computador descrito en el ejemplo anterior se analizará con más detalle en los Capítulos 8 y 9.

Como se indicó en el Capítulo 1, rara vez se programa en lenguaje máquina. En un lenguaje de alto nivel, como en C o Java, con dos instrucciones se hubiese podido redactar el

programa anterior, no siendo necesario hacer manualmente una **asignación de memoria** para el programa ni de registros para los datos, ya que esto lo hace automáticamente el traductor y/o el sistema operativo.

Se ha incluido este programa con el único objetivo de poner de manifiesto cómo funciona la Unidad de Control, y se interrelacionan las distintas unidades que constituyen el computador. Los detalles más significativos se analizan en las siguientes secciones.

7.3 El procesador

De lo indicado en las dos secciones anteriores se deduce claramente que el procesador es la unidad que determina el funcionamiento de un computador. Esta sección se dedica expresamente a dicha unidad. En primer lugar (Sección 7.3.1) se describen los elementos internos de un procesador, algunos de los cuales han sido introducidos previamente de manera escueta. Posteriormente (Sección 7.3.2) se analiza la forma en que la unidad de control descompone una instrucción máquina en microoperaciones, que se ejecutan en sucesivos ciclos de reloj, y se trata aparte el caso singular de las instrucciones de control (Sección 7.3.3), que permiten alterar el orden secuencial de ejecución de un programa.

7.3.1 Elementos internos de un procesador

La Figura 7.9 muestra algunos detalles internos de un ejemplo de procesador, que son de gran importancia. La interconexión entre estos elementos no se explicita completamente ya que depende del computador concreto, y se analizará más adelante (Capítulos 9 y 12).

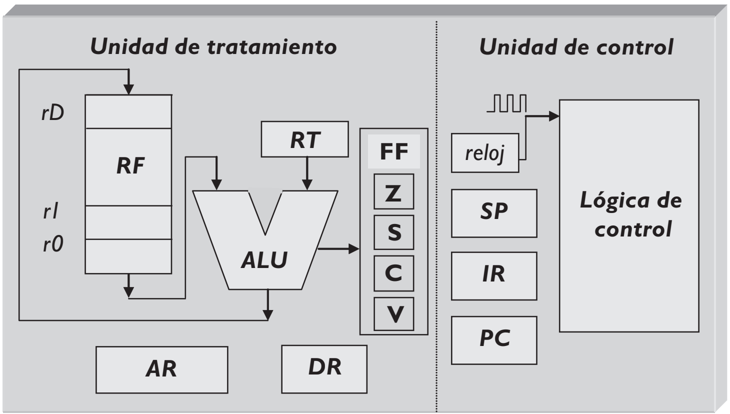


Figura 7.9. *Elementos básicos de un procesador.*

El procesador dispone de un conjunto de registros, denominados **registros de uso general** (*r0* a *rD*, en el ejemplo de la Figura 7.9), que suelen estar integrados en un **bloque de registros** (también denominado **archivo de registros**, ***RF***, *Register File*). Estos registros se utilizan como almacén temporal de los datos con los que va a operar la ALU o de resultados intermedios. También pueden dedicarse a almacenar direcciones de memoria. La longitud de estos registros suele ser la de una palabra, aunque es corriente que existan registros de distinta longitud; así, en un computador de 32 bits, es habitual que se puedan utilizar registros de media palabra (16 bits), de palabra (32 bits) y de doble palabra (64 bits). A veces los lenguajes máquina incluyen instrucciones para operar entre un registro significado,

denominado **acumulador** (*A*) y otro registro o una posición de memoria, depositándose el resultado en el acumulador. En el ejemplo de la Figura 7.9 no se ha incluido un acumulador; no obstante hay un registro, denominado **registro temporal** (***RT***), para almacenar en él uno de los operandos de la ALU. En definitiva, del esquema de la figura se deduce que la ALU opera con el contenido de uno de los registros de *RF* y el contenido de *RT*, guardando el resultado en uno de los registros de *RF*.

Asociado a la ALU hay unos **biestables indicadores** o **biestables de condición** (en inglés, *flag flip-flops;* ***FF***, en la figura), que se ponen a 1 ó 0 dependiendo de la última operación realizada en la ALU. Así, es normal que existan, biestables tales como:

C: Acarreo.
S: Indicador de signo (si el último resultado de la ALU es negativo, por ejemplo $S \leftarrow 1$).
Z: Indicador de cero (si el último resultado de la ALU es cero, $Z \leftarrow 1$).
P: Indicador de paridad, se pone a 1 si la paridad del resultado es par.
V: Indicador de desbordamiento, etc.

Según vimos anteriormente (Figura 7.6), la memoria principal y los periféricos se conectan con el procesador por medio de dos buses; uno de direcciones y otro de datos; además de con las dos señales de control $R/\overline{W}$ e $IO/\overline{M}$. Para un buen funcionamiento del conjunto, el procesador dispone de un **registro de dirección** (***AR***, *Address Register*), donde deberá ubicarse la dirección del dato/instrucción a leer o escribir, y un **registro de datos** (**DR**, *Data Register*), donde se almacenará el dato a escribir en la memoria o en un puerto de salida, o la información leída de la memoria o de un puerto de entrada, dependiendo del caso (Figura 7.10).

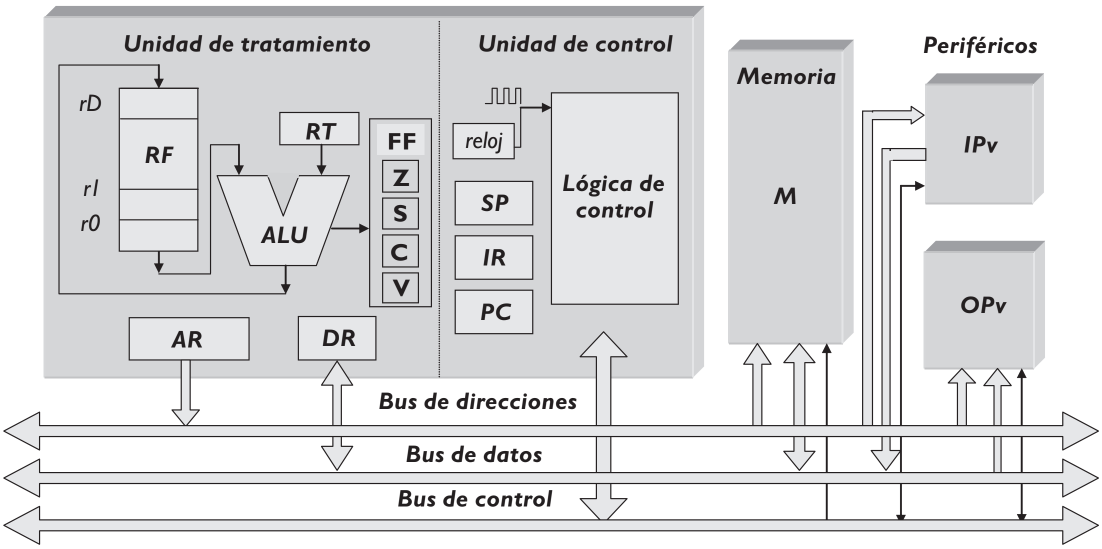


Figura 7.10. *Interconexión unibús del procesador con la memoria y los puertos de E/S.*

Como se ha indicado con anterioridad el procesador o CPU está formada por la Unidad de Control y la Unidad de Tratamiento (Figura 7.9). La Unidad de Control, contiene la **lógica de control**, que está constituida por los circuitos que generan las distintas señales de control, y el **reloj**, que es un generador de pulsos, con los que se sincronizan todas las microoperaciones que implican la ejecución de las distintas instrucciones máquina. También en la Unidad de

Control se encuentra el **registro de instrucción** (***IR***, *Instruction Register*), que está dedicado a memorizar temporalmente la instrucción del programa que la unidad de control está interpretando o ejecutando. Recuérdese (Sección 7.2) que el programa a ejecutar reside en la memoria principal, y la unidad de control va *captando* o *buscando las instrucciones* secuencialmente de la memoria principal, para poder interpretarlas y generar las señales de control. La captación de instrucción implica leer la instrucción de memoria y almacenarla en el registro de instrucción. Pero para captar la instrucción, el procesador debe, de alguna forma, saber el punto del programa que está en ejecución o, en otras palabras, contabilizar la dirección de memoria donde se encuentra la instrucción en ejecución. Este es el objetivo del ***contador del programa***[1] (***PC***, *Program Counter*), que es un registro-contador, que actúa como contador ascendente binario, con posibilidad de carga paralela (Sección 7.1), y que *contiene la dirección de memoria dónde se encuentra la instrucción siguiente a ejecutar.*

El último elemento que queda por describir es el **puntero de pila** (***SP***, *Stack Pointer*). El puntero de pila físicamente es, como el contador de programa, un contador binario ascendente con la opción de carga en paralelo. Su cometido está relacionado con una estructura pila (LIFO) que se mantiene en la memoria principal relacionada con las llamadas a (y retornos de) subrutinas, y que se analizarán en detalle en la Sección 7.3.3.

7.3.2 Temporización en la ejecución de instrucciones

Para iniciar la ejecución de un programa se ubica en el *PC* la dirección de memoria donde comienza dicho programa. El contenido de *PC* se transvasa a *AR* (esto, abreviadamente, se simboliza así: *AR*←*PC*), y la CU da a la memoria la orden de leer ($R/\overline{W}=1;\ IO/\overline{M}=0$). Después de un tiempo determinado (**tiempo de acceso a memoria)**, en el bus de entrada/salida de memoria, aparecerá la información contenida en la posición de dirección *PC*; es decir, la instrucción del programa (en este caso la primera), cargándose en el registro *DR* (*DR*← *M(AR)*). Posteriormente la información contenida en *DR*, esto es, la instrucción, se carga en el registro *IR* (*IR*← *DR*). También el contenido del contador de programa es incrementado (*PC*←*PC+1*), apuntando así a la siguiente instrucción (por simplificar, en esta descripción se supone que cada instrucción ocupa una sola posición de memoria).

Una vez cargada la instrucción en *IR* es decodificada y ejecutada bajo la monitorización de la CU. Si, por ejemplo, la instrucción máquina en ejecución es *sumar el contenido de dos registros con un dato y dejar el resultado en otro*, la CU genera las señales de control necesarias para llevar el primer operando al registro *RT* (Figura 7.9), llevar el contenido del otro operando a la otra entrada de la ALU, realizar la suma y llevar el resultado de la ALU al registro indicado.

Después de ejecutada la instrucción en curso, la CU vuelve a repetir el ciclo anterior; es decir, capta una nueva instrucción (la lleva a *IR*) y después la decodifica y la ejecuta. Este ciclo se repite iterativamente hasta que concluye la ejecución del programa.

Resumiendo lo dicho en los párrafos anteriores, *el contador de programa siempre contiene la dirección de memoria de la siguiente instrucción que se va a ejecutar.* Por tanto, una vez leída una instrucción el valor de *PC* debe incrementarse en 1, para *apuntar* a la siguiente instrucción. Todas las instrucciones comienzan siempre con una **fase de captación de instrucción**. Después tiene lugar una **fase de ejecución**, que es específica del código

[1] Con frecuencia al contador de programa también se le denominan **puntero de instrucciones** (**IP**, *Instruction Pointer*).

de operación (**codop**) de cada instrucción, y en ella se activan los distintos elementos de la unidad de tratamiento necesarios para la ejecución, propiamente dicha, de la instrucción. En definitiva la CU genera las señales de control necesarias para realizar las siguientes operaciones (Figura 7.11):

- **Fase de captación de instrucción:**

  $AR \leftarrow PC$
  $DR \leftarrow M(AR)$
  $IR \leftarrow DR$
  $PC \leftarrow PC+1$

  (Suponemos que cada instrucción ocupa una sola posición de memoria.)

- **Fase de ejecución:** se realizan las operaciones específicas correspondientes al **codop** de la instrucción almacenada en IR. Dependiendo de la instrucción se efectúan operaciones tales como captación o búsqueda de operandos en memoria, cálculos en la ALU, almacenamiento de resultados en registros o memoria, etc.

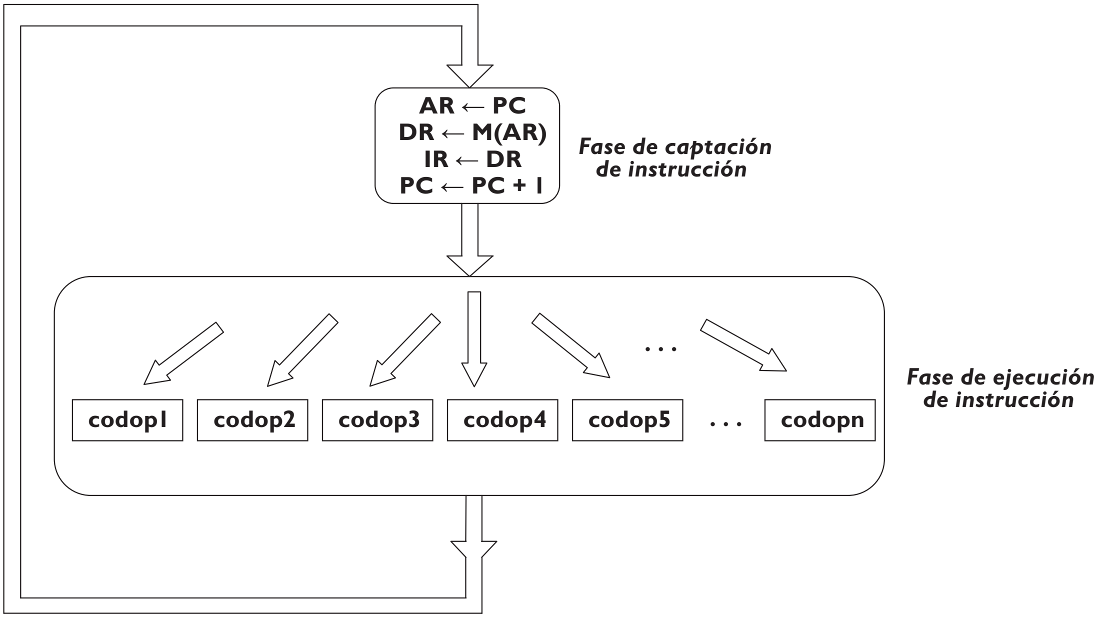


Figura 7.11. *Diagrama que muestra el ciclo y microoperaciones generadas por la unidad de control para ejecutar una instrucción máquina.*

De la descripción anterior se deduce que la ejecución de una instrucción lleva consigo, por lo menos, un acceso a memoria (para captar la instrucción), pudiendo efectuar accesos adicionales (captación de datos, memorización de resultados, etc.).

El tipo de operaciones indicadas anteriormente (carga de un registro: $AR \leftarrow PC$, $IR \leftarrow DR$; lectura de memoria: $DR \leftarrow M(AR)$; incremento del contador de programa: $PC \leftarrow PC+1$, etc.) son las acciones más elementales que puede hacer el computador, y reciben el nombre de **microoperaciones**. Se puede concluir que una *instrucción máquina*, implica la realización de un conjunto determinado de *microoperaciones* en un orden preestablecido.

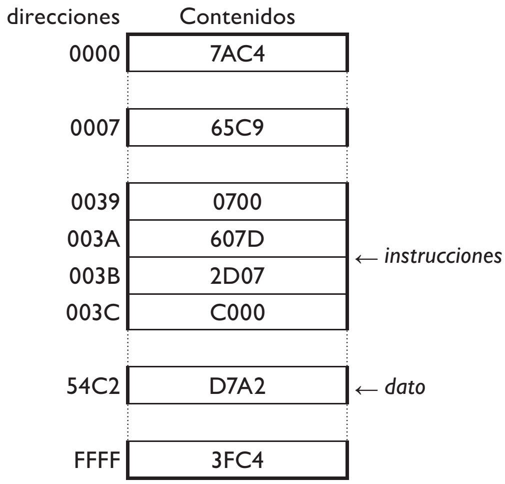


Figura 7.12. *Contenidos de memoria para los Ejemplos 7.2, 7.3 y 7.4.*

EJEMPLO 7.2

Supóngase que se dispone de un computador organizado en palabras de 16 bits. En la posición H'0039 de memoria (Figura 7.12) se encuentra la instrucción H'0700 que, simbólicamente se representa como LD r7,(rD), y carga el contenido del registro r7 con el dato que se encuentra en la posición de memoria que especifica rD; es decir:

r7 ← M(rD)

Suponiendo que en rD se encuentra el valor H'54C2, y los contenidos de memoria de la Figura 7.12, indicar las microoperaciones que se realizarán durante la ejecución de la instrucción, y los cambios que van teniendo lugar en PC, IR, AR, DR y R7.

Tabla 7.3. *Microoperaciones y contenidos de los registros durante la ejecución de la instrucción LD r7,(rD).*

		Contenidos de registros					
Fase	**Microperacioón**	**PC**	**IR**	**AR**	**DR**	**R7**	
		0039	-	-	-	-	← *Valores iniciales*
Captación de instrucción	AR ← PC	0039	-	***0039***	-	-	
	DR ← M(AR)	0039	-	0039	***0700***	-	
	IR ← DR	0039	***0700***	0039	0700	-	
	PC ← PC+1	***003A***	0700	0030	0700	-	
Ejecución de instrucción	AR ← rD	003A	0700	***54C2***	0700	-	
	DR ← M(AR)	003A	0700	54C2	***D7A2***	-	
	r7 ← DR	003A	0700	54C2	D7A2	***D7A2***	

EJEMPLO 7.2 (Cont.)

Teniendo en cuenta el objetivo de la instrucción y la Figura 7.10 y la Figura 7.12, las microoperaciones que efectuará el computador son las que se muestran en la Tabla 7.3. Se observa que para ejecutar la instrucción máquina del ejemplo se necesitan 7 microoperaciones que genera la unidad de control. La instrucción realiza dos accesos a memoria: uno para captar la instrucción, y otro para captar el dato a almacenar en R7.

EJEMPLO 7.3

Supóngase ahora, que en la posición H'003A de memoria (Figura 7.12) se encuentra la instrucción H'607D que, simbólicamente se representa como ADDS r0,r7,rD, que suma (en notación complemento a 2) los contenidos de r7 y rD y almacena el resultado en r0; es decir:

r0 ← r7+rD

Indicar las microoperaciones que se realizarán durante la ejecución de la instrucción, y los cambios que van teniendo lugar en PC, IR, AR, DR y R0.

Teniendo en cuenta el objetivo de la instrucción y las Figuras 7.10 y 7.12, las microoperaciones que efectuará el computador son las que se muestran en la Tabla 7.4. Se observa que para ejecutar la instrucción máquina del ejemplo se necesitan 6 microoperaciones que genera la unidad de control, y sólo se hace un acceso a memoria para captar la instrucción.

Al final de la ejecución de la instrucción los biestables indicadores pasan a tener los siguientes valores:

C=1; S=0; Z=0; P=0; V=0

Tabla 7.4. *Microoperaciones y contenidos de los registros durante la ejecución de la instrucción LD r7,(rD).*

Fase	Microoperacioón	Contenidos de registros				
		PC	IR	AR	DR	R7
		003A	0700	54C2	D7A2	-
Captación de instrucción	AR ← PC	003A	0700	***003A***	D7A2	-
	DR ← M(AR)	003A	0700	003A	***607D***	-
	IR ← DR	003A	***607D***	003A	607D	-
	PC ← PC+1	***003B***	607D	003A	607D	-
Ejecución de instrucción	RT ← rD	003B	607D	003A	607D	-
	r0 ← r7+RT	003B	607D	003A	607D	***2C6F****

← Valores iniciales

* Debido a que D7A2+54C=2C6F.

En la Figura 7.13 se indican esquemáticamente, dentro del ciclo de ejecución de la unidad de control, las microoperaciones a realizar con las instrucciones de carga de registro y de suma. Las microoperaciones correspondientes a otras instrucciones máquina de tipo aritmético o lógico son muy similares.

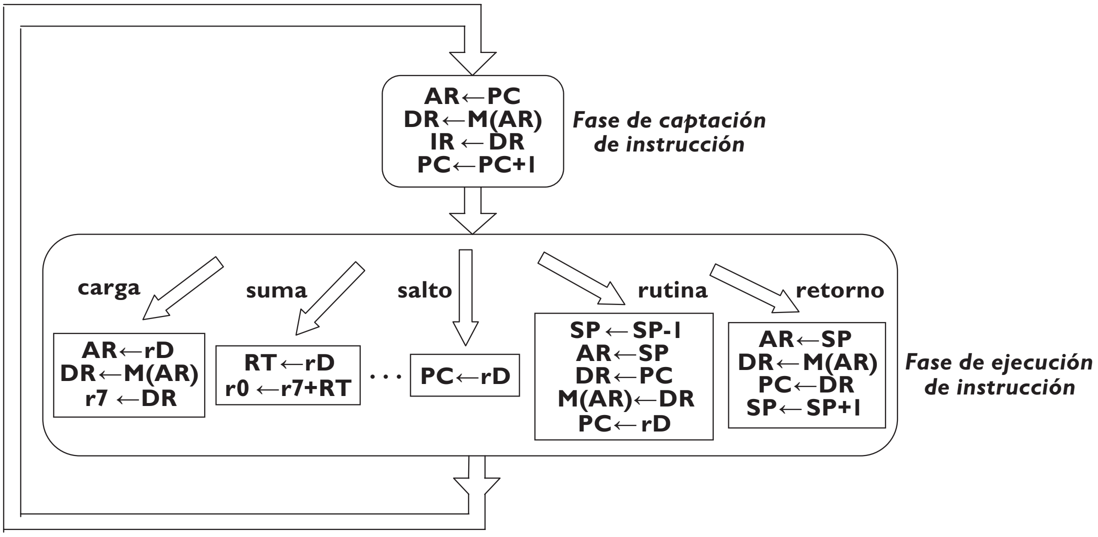


Figura 7.13. *Diagrama mostrando las microoperaciones que tienen lugar cuando se ejecutan las instrucciones de carga de registro, suma, salto, llamada a subrutina y retorno de subrutina.*

7.3.3 Gestión de las instrucciones de control

Según lo visto anteriormente el contador de programa, *PC*, durante la fase de captación de instrucción, se incrementa en 1, apuntando a la siguiente instrucción a ejecutar. Se plantea el problema de la ejecución de las instrucciones de control de flujo (Sección 1.4), ya que con el esquema anterior el programa se va ejecutando secuencialmente, según el orden en que están las instrucciones almacenadas en la memoria principal, y las instrucciones de bifurcación precisamente permiten alterar ese orden de ejecución.

Conviene distinguir entre dos tipos de instrucciones de control de flujo:

1. Bifurcaciones (o saltos condicionales) y saltos (incondicionales).
2. Llamadas a procedimientos y retornos de procedimientos, condicionales o no.

A continuación se describen estos dos tipos de instrucciones.

Instrucciones de control de flujo que no hacen referencia a procedimientos

Con este tipo de instrucciones se puede alterar el orden de ejecución de un programa, saltando a una instrucción ubicada en una dirección de memoria arbitraria, *ds*, especificada en la instrucción de salto. Una vez ejecutada la instrucción de salto, el programa ejecuta la instrucción contenida en la posición *ds* y las que se encuentren sucesivamente a partir de ella (hasta el fin del programa o hasta otra alteración del flujo de control provocada por otra instrucción de este tipo). La dirección de salto, dependiendo del juego de instrucciones del procesador, puede darse de distintas formas; por ejemplo puede estar en la propia instrucción (direccionamiento directo), o puede estar contenida en un registro del procesador (direccionamiento indirecto).

Supóngase, por ejemplo, que en las posiciones de memoria *1A2E, 1A2F, 1A30* se tuviese la instrucción de salto *BR 32B7*. Esta instrucción significa bifurcar ("*branch*") a la dirección *32B7*. Cuando finaliza la ejecución de la instrucción citada, en vez de ejecutarse la siguiente instrucción que esté en memoria (en la posición *1A31*), se ejecutaría la que estuviese en la posición *32B7* y siguientes.

La implementación de este tipo de instrucciones por el procesador es muy sencilla: en su fase de ejecución el procesador cambia el contenido del contador de programa por *ds* –en el ejemplo *ds* es 32B7– (recuérdese que PC físicamente es un contador con posibilidad de *carga en paralelo*):

$$PC \leftarrow ds$$

De esta forma, en la siguiente captación de instrucción, al leer de memoria la instrucción cuya dirección está en PC (como siempre, Figura 7.13), lo que se capta es la instrucción que está en la dirección *ds,* en otras palabras, se produce automáticamente el salto.

En el caso del procesador que estamos considerando como ejemplo a lo largo del capítulo, suponemos que la dirección de salto se encuentra en *rD*, con lo que la microoperación a realizar será:

$$PC \leftarrow rD$$

tal y como se indica en la Figura 7.13.

En las instrucciones de **salto condicional**, el salto se produce sólo si se cumple alguna o algunas condiciones, establecidas por el valor de alguno de los biestables indicadores o algún registro. Por ejemplo, la instrucción *BZ 32B7*, hace que se salte a ejecutar la instrucción que está a partir de *32B7* (y sucesivas) si el biestable *Z* está a *1.*

Instrucciones de control de flujo que hacen referencia a subrutinas

Una llamada a una **subrutina** (también denominada **rutina**, **subprograma** o **procedimiento**) es una ruptura de la secuencia normal de ejecución de las instrucciones de un programa, de forma que tras la ejecución de la *instrucción de llamada*, se ejecuta otro programa, denominado, subrutina, rutina, subprograma o procedimiento. Una vez ejecutada la subrutina, se retorna (por medio de una *instrucción de retorno*) al programa desde el que se hizo la llamada, que continúa ejecutándose a partir de la posición desde la que se saltó. A su vez, una subrutina puede incluir llamadas a otras subrutinas, y así sucesivamente (Figura 7.14).

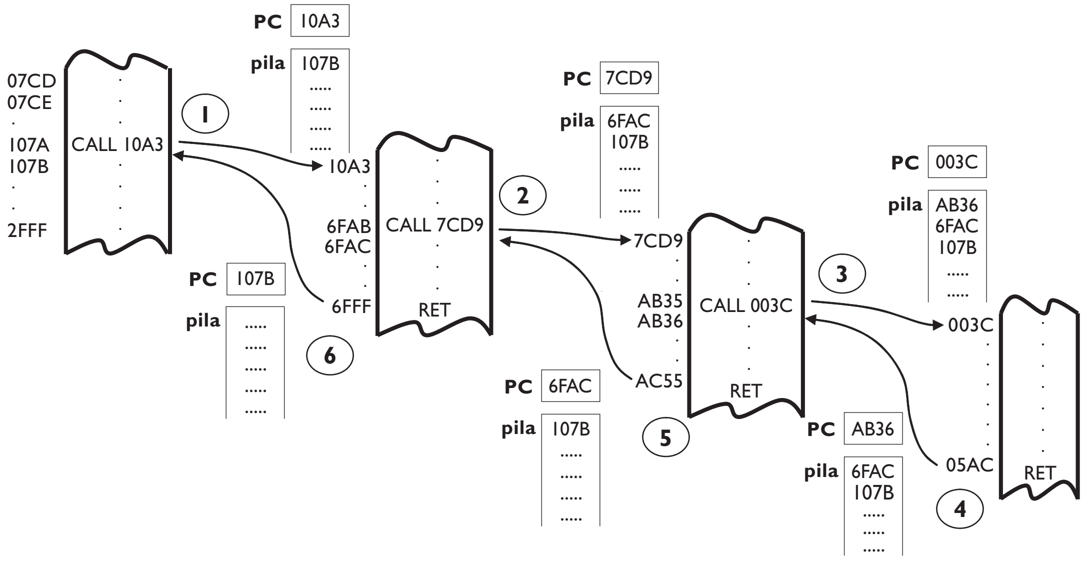


Figura 7.14. *Ejemplo de cómo se gestionan las llamadas y retornos de los procedimientos.* ***CALL ds*** *es la instrucción de llamada y* ***RET*** *la de retorno.*

Desde el punto de vista de la arquitectura del procesador, la diferencia entre una bifurcación o salto y una llamada a una subrutina, es que cuando acaba de ejecutarse la subrutina llamada hay que *retornar* al programa o subrutina que lo llamó, concretamente se debe retornar a la instrucción inmediatamente después de la de llamada, que es precisamente la que se encuentra en la dirección contenida en el *PC* al ejecutarse la instrucción de llamada. Por tanto, habrá que memorizar temporalmente *los contenidos del PC de las instrucciones de llamada* a subrutinas, para recuperarlas en el *PC,* en el momento oportuno. Cuando se retorna a una subrutina de llamada, la dirección de vuelta puede eliminarse de la memoria. El almacenamiento de las direcciones de las instrucciones de llamada se realiza en una memoria o estructura de datos de tipo **pila** (**memoria LIFO**), véase Secciones 2.2.4 y 6.7.1. En la Figura 7.14 puede verse un ejemplo de tres llamadas sucesivas a subrutinas. La secuencia de hechos relacionados con dichas llamadas es la siguiente:

1) Al ejecutar la instrucción del primer procedimiento que está en la posición 107A se llama a la subrutina que comienza en la dirección 10A3. La UC almacena el valor del *PC* en ese momento (que es 107B) en la cabecera de la pila, y cambia el contenido de *PC* por la dirección de salto: 10A3. De esta forma, después de la instrucción en 107A se ejecuta la instrucción de 10A3 y sucesivas.
2) Llega la ejecución de la instrucción en 6FAB, que es una llamada a la subrutina que empieza en la dirección 7CD9. La UC almacena el valor del *PC* en ese momento (que es 6FAC) en la pila, y cambia el contenido de *PC* por la dirección de salto: 7CD9. De esta forma, después de la instrucción 6FAB se ejecuta la instrucción de 7CD9 y sucesivas.
3) Llega la ejecución de la instrucción de AB35, que es una llamada a la subrutina que empieza en la dirección 003C. La UC almacena el valor del *PC* en ese momento (que es AB36) en la pila, y cambia el contenido de *PC* por la dirección de salto: 003C. De esta forma, después de la instrucción AB35 se ejecuta la instrucción de la posición 003C y sucesivas.
4) Llega la ejecución de la instrucción que se encuentra en 05A4, que es un retorno de subrutina. La UC extrae el valor almacenado en la cabecera de la pila, que es la última dirección almacenada (AB36), y lo carga en el *PC*. De esta forma a continuación se ejecutará la instrucción AB36 y sucesivas; es decir, se ha producido el retorno al procedimiento que llamó al que acaba de finalizar. Al efectuar la extracción de la cabecera de la pila (valor AB36), este valor desaparece de la pila, pasando 6FAC a ser la nueva cabecera.
5) Llega la ejecución de la instrucción en AC55, que es un retorno de subrutina. La UC extrae el valor almacenado en la cabecera de la pila, que en este caso es 6FAC, y lo carga en el PC. De esta forma, a continuación, se ejecutará la instrucción en 6FAC y sucesivas; es decir, se ha producido el retorno al procedimiento que llamó al que acaba de finalizar.
6) Llega la ejecución de la instrucción en 6FFF, que es un retorno de subrutina. La UC extrae el valor almacenado en la cabecera de la pila, que en este caso es 107B, y lo carga en el *PC*. De esta forma, a continuación, se ejecutará la instrucción 107B y sucesivas; es decir, se ha producido el retorno al procedimiento que llamó al que acaba de finalizar.

Nótese que la recuperación de las direcciones de retorno ha de hacerse en orden inverso al de su almacenamiento, y éste es precisamente el motivo para utilizar una estructura de tipo LIFO para almacenarlas.

De acuerdo con lo anterior, las microoperaciones a realizar durante la *fase de ejecución* de las instrucciones relacionadas con las llamadas a subrutinas serán:

Llamada a procedimiento

$$Pila \leftarrow PC$$
$$PC \leftarrow ds$$

donde, en este caso, *ds*, es la dirección de comienzo de la subrutina, y que hemos supuesto viene especificada como parte de la propia instrucción de llamada.

Retorno de procedimiento

$$PC \leftarrow Pila$$

Obsérvese que para producir el retorno, lo único que debe hacer la CU es hacer que la dirección almacenada en la cabecera de la pila (que es la dirección de retorno) se transfiera al PC. La siguiente instrucción que se capte, por lo tanto, será ya del programa que llamó al procedimiento.

En los computadores de muy alta velocidad la memoria pila suele diseñarse con circuitos específicos **(pila hardware**, véase Sección 6.7.1), y forma por tanto parte del procesador; pero, en la mayor parte de computadores, la pila se gestiona o *simula* en la memoria principal (**pila software**). En efecto, *las direcciones de retorno* (los contenidos de *PC* en las instrucciones de llamada) se almacenan en una región de la memoria principal que se denomina **zona LIFO**. El programador de lenguaje máquina (o el sistema operativo o el traductor) determinan la zona de memoria que se reservará como memoria LIFO para almacenar las direcciones de retorno.

El procesador contiene en su interior un registro específico, donde la UC almacena la *dirección* de la zona LIFO donde se guardó la última dirección de retorno. Este registro se denomina **puntero pila** o ***SP*** (de *Stack Pointer*) (véase Figura 7.9). Cuando se ejecuta una instrucción de retorno, la CU ordena leer la última dirección almacenada en la pila; es decir, *el contenido de la dirección de memoria cuya posición está en SP*. Este contenido se cargará en el contador de programa (*PC*), concluyendo con ello la ejecución de la instrucción de retorno.

Antes de efectuar llamada a procedimiento alguno debe cargarse en *SP* un valor inicial (la dirección límite de la zona LIFO). Existen instrucciones máquina específicas para efectuar esta carga. El procesador automáticamente se encarga de almacenar y recuperar las direcciones de retorno, según se vayan ejecutando instrucciones máquina de llamada y retorno de procedimientos.

En el ejemplo de procesador que estamos analizando en este capítulo las direcciones de salto se encuentran en el registro *rD.* Para implementar la llamada a una subrutina, primero hay que decrementar SP para que apunte a la siguiente posición libre de la memoria, que pasará a ser la nueva cabecera de la pila (Figura 7.15), después hay que salvar en la posición *SP* de memoria el valor del contador de programa (para lo cual hay que llevar *SP* a *AR* y *PC* a *DR*) y, después debe llevarse la dirección de salto (que está en *rD*) a *PC*. En definitiva, las microoperaciones a realizar en una instrucción de llamada a subrutina son:

$$SP \leftarrow SP{-}1$$
$$AR \leftarrow SP$$
$$DR \leftarrow PC$$
$$M(AR) \leftarrow DR$$
$$PC \leftarrow rD$$

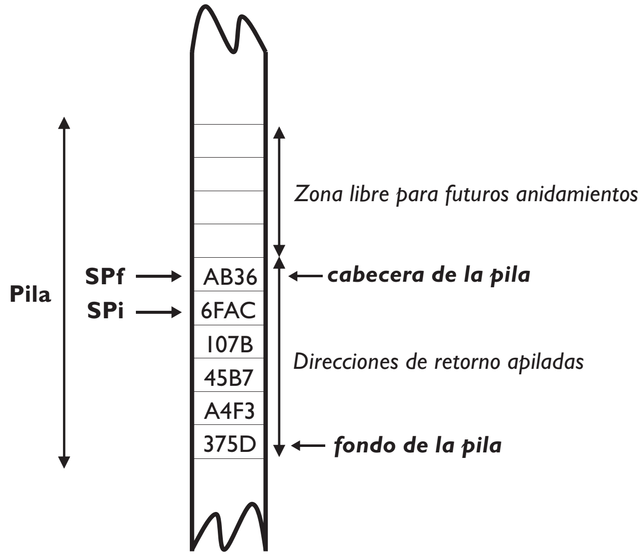


Figura 7.15. *Esquema que muestra cómo se gestiona la pila en la memoria principal.* ***SPi*** *y* ***SPf*** *representan, respectivamente, las direcciones contenidas en SP antes y después de hacer la llamada a la rutina.*

Siempre que se llama a una subrutina se realizan las microoperaciones que se acaban de describir, de forma que la zona LIFO se va rellenando, en la Figura 7.15, hacia arriba (desde direcciones altas hacia direcciones bajas), y en SP se conserva la posición donde se encuentra la dirección de retorno de la rutina en ejecución.

En las ejecuciones de retornos de procedimientos el puntero se va incrementando (la zona LIFO se va vaciando, en la Figura 7.15 hacia abajo, hacia las direcciones altas). En el procesador que estamos considerando, la CU genera las señales de control que efectúan las siguientes microoperaciones (Figura 7.13):

$$AR \leftarrow SP$$
$$DR \leftarrow M(AR)$$
$$PC \leftarrow DR$$
$$SP \leftarrow SP+1$$

recuperándose así en el *PC* la dirección de retorno del procedimiento que estaba ejecutándose. Una nueva llamada a un procedimiento destruiría (sin causar ningún problema por no necesitarse ya) la última dirección de retorno recuperada.

Si se previese que como máximo se produjesen 64 llamadas sucesivas entre subrutinas (es decir, nivel de anidamiento 64), se tendrían que reservar 64 posiciones de memoria como zona LIFO, y antes de llamar a cualquier procedimiento habría que cargar *SP* con el valor de la frontera de direcciones más alta de dicha zona.

Como la gestión de pila en la memoria principal implica la realización de lecturas o escrituras en la memoria principal, esta forma de implementación de la pila es mucho más lenta que utilizar una pila hardware específica. No obstante, el segundo tipo de implementación es menos versátil (el nivel de anidamiento viene fijado por el tamaño físico de la pila) y el tamaño del procesador será mayor (en vez de contener un sólo registro, SP, debe contener

toda la pila). Los procesadores actuales de altas prestaciones suelen optar por una solución mixta: tienen pila hardware y puntero de pila, de forma tal que cuando se desborda la pila hardware, las direcciones de retorno se almacenan en la memoria principal.

7.4 El concepto de interrupción

Una **interrupción** es una detención de la ejecución de un programa antes de que concluya para ejecutar otro. Normalmente, la detención se realiza para ejecutar otro procedimiento prioritario denominado **programa de gestión de la interrupción**, soliéndose continuar después con la ejecución del programa interrumpido.

Las interrupciones pueden ser provocadas por eventos externos o internos (en este último caso se denominan excepciones), y sus orígenes pueden ser muy diversos; por ejemplo:

- **Fallo de la alimentación.** Por ejemplo, los PC portátiles suelen incluir un circuito detector de la carga de los acumuladores de energía eléctrica. Cuando se detecta que la carga está por debajo de un determinado nivel, y por lo tanto hay riesgo de que se pierdan los trabajos que estén en la memoria RAM, se le hace entrar en un estado que se suele denominar **suspensión** o **hibernación**. Al descender la carga de la batería por debajo del nivel preestablecido, se produce una interrupción cuya rutina de servicio avisa al usuario de esta contingencia y salva todo el contenido de la memoria en disco. De esta forma el usuario, una vez recargados los acumuladores o establecida la alimentación a través de la red, puede continuar trabajando con el PC en el estado en que se encontraba cuando se detecto el fallo de alimentación.
- **Anomalías del hardware.** En el caso de que se detecte un fallo del hardware, puede generarse una interrupción que lance a ejecutar un programa que produzca el consiguiente mensaje de error y/o reconfigure el hardware. Por ejemplo, si en la lectura de un DVD se detecta que éste no está adecuadamente insertado en la unidad correspondiente, el procedimiento de atención de esta interrupción enviará un mensaje a la pantalla.
- **Petición de E/S de un periférico.** El método más común (véase Sección 9.5.4.2 para más detalles) de efectuar una operación de E/S es por medio de interrupciones. Por ejemplo, en la mayor parte de sistemas, cuando estamos escribiendo en un teclado, cada vez que pulsamos una tecla se solicita una interrupción para que se detenga el programa que pudiese estar en ejecución y se lance una rutina del sistema operativo que capta la información tecleada.
- **Desbordamiento.** Cuando en un cálculo aritmético se produce un desbordamiento es normal producir una excepción para lanzar a ejecución un programa que dé un mensaje de error, o se tomen otras medidas tendentes a no falsear los resultados de la ALU.
- **Utilización de una instrucción máquina inexistente.** Esta anomalía se produciría cuando en el registro de instrucción (IR) se carga un valor que no corresponde a ningún código del lenguaje máquina. El procedimiento que atendería a la interrupción daría un mensaje de error.
- **Gestión de tiempo compartido.** El tiempo compartido (Sección 13.5.1) permite la ejecución concurrente de varios programas ubicados simultáneamente en memoria. El sistema operativo va dando pequeños turnos sucesivos de tiempo a cada uno de los programas, de forma tal que todos van avanzando en su ejecución; ahora bien, en cada instante el procesador sólo puede atender a un programa. Los sistemas con esta posibilidad disponen de un temporizador (denominado **reloj de tiempo real**) que genera una

señal de interrupción cuando transcurre el periodo de tiempo asignado a cada programa. La atención de la interrupción consiste en la suspensión momentánea del programa en curso y la continuación del programa siguiente en cola. Posteriormente, cuando se hayan consumido los turnos correspondiente a los demás programas disponibles para ejecución en la memoria, se reanuda la ejecución del programa interrumpido.

- **Violación de la protección de memoria.** Es corriente que en la memoria principal se encuentren ubicados diversos programas (un ejemplo es el caso del tiempo compartido, descrito en el párrafo anterior). El computador dispone de mecanismos para detectar si, por error, un programa invade la zona de memoria donde se encuentra otro programa. Cuando se produce esta situación se genera una excepción que lanza a ejecutar un programa que gestiona adecuadamente el problema.
- **Peticiones del programador o usuario.** Existe la posibilidad de que el propio programador o usuario deseen provocar una interrupción. Esto ocurre, por ejemplo, en los programas de rastreo o seguimiento de ejecución de instrucciones. Si un error de un programa bloquea un computador, el usuario puede provocar una interrupción desde el teclado (en un computador personal, tecleando Ctrl-C, o Ctrl-Alt-Del, por ejemplo).

La producción de una interrupción tiene implicaciones hardware y software, como se comprende al analizar los pasos que habitualmente se siguen en la gestión de una de ellas. En el caso más general, de que después de atender una interrupción haya que continuar con el programa interrumpido, tienen lugar los siguientes pasos (Figura 7.16):

1) **Petición de interrupción** *(“Interruption Request”*, o IR). La petición de interrupción se realiza por medio de una señal eléctrica enviada al procesador (es una *señal de estado* de las descritas en la Sección 1.2). Por lo general el procesador no atiende inmediatamente la petición de interrupción, si no que antes acaba de ejecutar la instrucción o ciclo de lectura/escritura en memoria en curso. Un mismo procesador dispone de varias líneas de petición de interrupción. Por medio de instrucciones adecuadas se pueden **enmascarar** (inhibir) selectivamente las distintas peticiones. De esta forma si se está ejecutando un procedimiento muy importante (salvar el contenido de memoria a causa de un fallo hardware, por ejemplo) pueden inhibirse otras interrupciones para que no se interrumpa la ejecución del procedimiento prioritario. Posteriormente, con instrucciones máquina específicas, se pueden habilitar las interrupciones o atender a las que estuviesen pendientes o hubiesen estado enmascaradas. Hay técnicas de arbitraje de peticiones de interrupción, de forma que cuando hay varias peticiones de interrupción pendientes se determina a cuál atender primero.
2) La **atención** o **tratamiento de la interrupción** (“*Interruption Acknowledge*” o IA) suele comenzar con un procedimiento de servicio de *inicio de la interrupción*. Este procedimiento analiza la causa de la interrupción (determina, por ejemplo, qué periférico la ha solicitado), inhabilita –si se requiere– la posibilidad de atender otras interrupciones, salva en memoria principal el contenido de los registros del procesador (incluyendo el estado de los biestables indicadores) que fuese a utilizar (y por tanto modificar) el procedimiento que atiende la interrupción, y pasa el control a este último. La salvaguarda en memoria que se hace de estos contenidos es para que no se pierdan con la ejecución del procedimiento preferente, y poder continuar posteriormente con el programa interrumpido. Normalmente, para mayor rapidez y comodidad, los contenidos se salvan en la memoria pila (la misma en que se salvan las direcciones desde donde se llaman a los procedimientos), teniendo el traductor o programador que tener

muy en cuenta el orden en que se apilan los distintos elementos, ya que la extracción de la memoria pila hay que hacerla en sentido inverso a la inserción (lo último que se inserta es lo primero que debe extraerse).

3) Se **ejecuta el procedimiento** o módulo **preferente**, atendiéndose, por tanto, al servicio requerido por la interrupción. Por ejemplo, si la interrupción la solicitó el periférico de entrada de código 1101, el procedimiento contiene las instrucciones necesarias para leer el contenido del puerto o puertos del mismo y enviar el mensaje correspondiente al usuario (posiblemente el cursor que aparece en pantalla).
4) Se ejecuta un **procedimiento de fin de interrupción** que, por ejemplo, restaura el contenido de los registros del procedimiento que se interrumpió (los extrae de la pila, caso de que se hubiesen almacenado allí), y habilita, en su caso, las interrupciones inhabilitadas en el Paso 2.
5) Se ***pasa el control al programa interrumpido,*** continuándose éste.

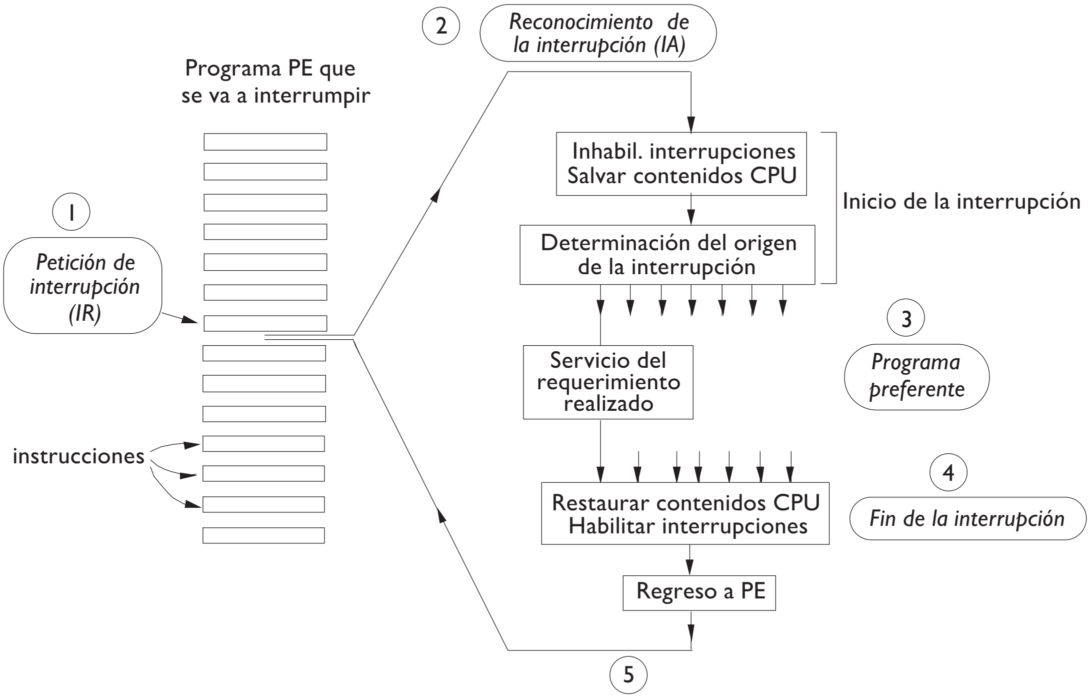


Figura 7.16. *Gestión de una interrupción de un procedimiento PE para atender a otro más preferente, volviéndose posteriormente a continuar ejecutándose el programa interrumpido.*

La rutina que atiende a la interrupción se suele denominar **Rutina de Servicio de Interrupción** (o **ISR**, "*Interruction Service Routine*"), y suele incluir los pasos 2, 3, 4 y 5 indicados anteriormente.

Puede observarse que la atención de una interrupción del tipo descrito es un caso muy particular de llamada a un procedimiento. De hecho un programa que interrumpe a otro puede a su vez ser interrumpido por otro (si no están enmascaradas todas las peticiones de interrupción), y así sucesivamente, gestionándose con facilidad la salvaguarda y restauración de los contenidos del contador de programa y de otros elementos del procesador de forma similar a como se gestionan las llamadas y retornos de procedimientos (Sección 7.1.3).

En un computador las interrupciones se producen con gran frecuencia. Para dar una idea de este hecho indicaremos que se ha observado que en un DEC-VAX 8800, con tiempo compartido y con una carga de trabajo normal, dos interrupciones sucesivas se producían con un intervalo de tiempo medio de 0,9 ms. [Hen93].

7.5 Conclusión

En este capítulo se ha efectuado una introducción a los computadores en el nivel de micro-máquina. Se han descrito las funciones que realizan los elementos internos de un procesador. También se ha presentado la temporización o ciclo que realiza el procesador para ejecutar las instrucciones máquina.

El concepto de interrupción es complejo pero básico dentro de los computadores actuales, por lo que lo hemos introducido en este capítulo. Concretamente hemos analizado las causas de producción de interrupciones y los protocolos de atención de éstas.

EJERCICIOS

7.1 Un procesador dispone, entre otros, de los siguientes elementos: registro de dirección (AR) de 32 bits, registro de datos (DR) de 16 bits, contador de programa (PC), y puntero de pila (SP). Indicar:
1) Número de hilos conductores del bus de datos.
2) Número de hilos conductores del bus de direcciones.
3) Tamaño máximo posible de la memoria principal (en MB o GB).
4) Tamaño en bits del registro PC.
5) Tamaño en bits del registro SP.

7.2 Suponga que el computador descrito en la Sección 7.3 dispone de la instrucción *memorizar,* ST r1, que almacena en la posición rD de memoria el contenido del registro r1. La instrucción tiene de código (en hexadecimal) 1100. Suponiendo que ésta instrucción se encuentra en la posición A777 de la memoria, que en rD se encuentra el valor 5ACD y en r1 FFFF, realice una tabla donde se indiquen las distintas microoperaciones que deben generarse durante la ejecución de la instrucción, los valores que tienen en cada momento los registros PC, AR, DR e IR, y los cambios producidos en la memoria.

7.3 Suponga que el computador descrito en la Sección 7.3 dispone de la instrucción *entrada* (IN r0,IP3) que almacena en el registro r0 el contenido del puerto de entrada IP3. El código (en hexadecimal) de la instrucción es 4003. Suponiendo que en un momento esta instrucción se encuentra en la posición de memoria A778 y que en el puerto IP3 se encuentra el valor FFFF, realice una tabla donde se indiquen las distintas microoperaciones que deben generarse durante la ejecución de la instrucción, los valores que tienen en cada momento los registros PC, AR, DR e IR, y los cambios producidos en el banco de registros.

7.4 Suponga que en la posición de memoria A779 del computador descrita en la Sección 7.3 se encuentra la instrucción de *salida H'*5105. Esta instrucción (*OUT OP5,r1)* lleva al puerto de salida OP5 el contenido del registro r1. Suponiendo que en un momento dado en r1 se encuentra el valor FFFF, realice una tabla donde se indiquen las distintas microoperaciones que deben generarse durante la ejecución de la instrucción, los valores que tienen en cada momento los registros PC, AR, DR e IR, y los cambios producidos en los puertos de salida.

7.5 Un computador tiene una memoria principal, organizada en Bytes, siendo su capacidad máxima de 32MB, y se quiere que el nivel de anidamiento máximo entre subrutinas sea de 256. Estando la memoria pila en las posiciones más altas de memoria, indicar en hexadecimal las posiciones de memoria concretas que habrá que reservar para la pila, suponiendo que ésta sólo se utilizará en las llamadas y retornos de subrutinas.

7.6 Se dispone de una memoria de 4 GBytes, organizada en palabras de 16 bits. El ancho de banda del bus de datos es de 150 Mbits/segundo. Obtener:
a) El ancho de los buses de datos y de dirección de la memoria.
b) Suponiendo que un proceso utiliza una pila para poder almacenar hasta 1.024 direcciones en las posiciones altas de memoria, indicar en hexadecimal cuáles serían las direcciones físicas reservadas para esa hipotética pila.
c) ¿Cuánto tiempo tardaría en transmitirse por el bus la pila completa?

7.7 Suponga que en el computador descrito en la Sección 7.3 en un momento dado el valor del puntero de pila es EFFF, y se hacen las llamadas a subrutinas que se indican resumida y ordenadamente en la tabla. Indicar, con un esquema similar al de la Figura 7.14, los cambios en el contador de programa, y cómo se iría rellenando y vaciando la pila en las llamadas y retornos, respectivamente.

Valor de rD en el momento de la llamada	Dirección donde se produce la llamada (CALL)	Dirección donde se produce el retorno (RET)
7CD4	AB00	*(programa principal)*
1F22	AC73	ACA9
7CD4	1FA5	1FCD
(no hay llamada)	AC73	ACA9

BIBLIOGRAFÍA

[Dow00] Dowsing, R. D.; Woodhams, F. W. D.; Marshall, I.: *Computers from logic to architecture;* 2nd Edt. MacGraw-Hill, 2000.

[Ham03] C. Hamacher; Vranesic, Z.; Zaky, S.: *Organización de computadores*, 5ª Ed., McGraw-Hill, 2003

[Hen93] Hennessy, J. L.; Patterson, D.: *Arquitectura de computadores. Un enfoque cuantitativo*. McGraw-Hill, 1993.

[Pri05] Prieto, A.; Prieto, B.: *Conceptos de Informática,* Serie Shaum, McGraw-Hill, 2005.

[Sta06] Stallings, W.: *Organización y arquitectura de computadores*. 7ª Edc. Prentice Hall, 2006.

[Tan00] Tanenbaum, A. S.: *Structured Computer Organization,* 5ª Edc. Prentice Hall, 2006.

CAPÍTULO 8

Lenguajes máquina y ensamblador

En este capítulo se describe un computador en el nivel de complejidad de **máquina convencional.** En este nivel el computador queda definido y descrito por su repertorio de instrucciones en lenguajes máquina y ensamblador. Estos lenguajes permiten prescindir (o reducir al máximo) la necesidad de conocer el computador a nivel de micromáquina y niveles inferiores.

Los lenguajes máquina y ensamblador (también denominados lenguajes de bajo nivel) están totalmente ligados a la estructura del computador. En primer lugar se describen los modos de direccionamiento básico, que indican las formas en las que el procesador capta los operandos o almacena los resultados. Después se dan unas ideas sobre los apartados o campos que suele contener una instrucción máquina (formatos). Posteriormente se hace la descripción a nivel de lenguaje máquina y ensamblador de un Computador Didáctico Elemental (que denominamos CODE-2), y que será diseñado en su totalidad en el Capítulo 9.

Aunque los lenguajes ensambladores son distintos de un computador a otro, la mayor parte de ellos sigue una misma filosofía, por lo que es de una gran utilidad utilizar una terminología común; esto es lo que pretende el estándar IEEE 694, que resumimos en una de las últimas secciones del presente capítulo. Consideramos de mayor interés describir esta normalización que analizar el lenguaje ensamblador de un procesador comercial concreto. Por último incluimos un análisis comparativo entre las características de los lenguajes máquina y ensambladores. Como material complementario a este capítulo, en la página web asociada a este libro se incluye un programa emulador de CODE-2 y un ensamblador cruzado utilizable en PC compatibles.

8.1 Modos de direccionamiento

Toda instrucción máquina de un computador dispone de un campo denominado **código de operación** (o abreviadamente, **codop**) que especifica la operación que realiza la instrucción. Además, la mayor parte de las instrucciones utilizan uno o varios campos donde se encuentra el operando u operandos, o información a partir de la cual pueden localizarse éstos (es decir, información con la que puede conocerse el registro o posición de memoria donde se encuentran los datos, o a donde hay que llevar un resultado). En las instrucciones de bifurcación la dirección a obtener, a partir del campo citado, es la correspondiente a la próxima instrucción a ejecutar (dirección a cargar en el contador de programa, PC).

Con frecuencia en la instrucción no se dan las **direcciones físicas de memoria** donde están los operandos, ya que ello obligaría a dedicar un gran número de bits de la instrucción para especificar la dirección; lo que se suele hacer es dar información a partir de la cual el procesador puede obtener aquélla. Piénsese que, por ejemplo, si de dispone de una memoria de 32 Mpalabras, solamente para direccionar físicamente una posición de ella la instrucción dedicaría 25 bits.

Los lenguajes máquina y ensamblador de cualquier procesador suelen disponer de gran diversidad de formas para especificar direcciones de memoria. Así, por ejemplo, el lenguaje máquina del microprocesador 68020 admite hasta 18 modos de direccionamiento, y el Pentium admite 9 [Sta06]. Un **modo de direccionamiento** (Fgura 8.1) describe la forma con que el procesador determina, durante la ejecución de una instrucción, el lugar donde se encuentra (o donde se almacenará) un operando o la dirección a la que hay que saltar (en el caso de una instrucción de bifurcación).

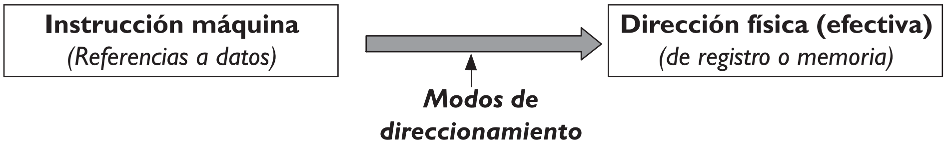


Figura 8.1. *El modo de direccionamiento establece la forma de obtener la dirección efectiva a partir de una referencia en un campo de dirección de una instrucción máquina.*

A continuación se describen los modos de direccionamiento más usuales. Frecuentemente una misma instrucción utiliza modos de direccionamiento distintos para sus diferentes operandos. En la Tabla 8.1 se muestran distintos ejemplos de los modos de direccionamiento que se verán a continuación, todos ellos dados conforme a la notación del estándar IEEE 694. Con frecuencia los ensambladores y los lenguajes de alto nivel permiten utilizar dentro de un mismo programa números en distintas bases de numeración; según el estándar citado, la base de un número se especifica con un prefijo de la forma siguiente:

B' número binario
D' número decimal
H' número hexadecimal
Q' número octal

Por ejemplo, H'7359, quiere decir que el número que se da a continuación (7359) está en base hexadecimal.

Tabla 8.1. *Ejemplos de tipos de direccionamiento (notación IEEE 694).*

Direccionamiento	Ejemplo	Explicación
Implícito	NOTC	Complementar bit de acarreo
Inmediato	SUB .0,#H'25	R0 7 R0 – 25h
Directo	SUB .0,/H'25A3	R0 7 R0 – M(25A3h)
Indirecto con autopostincremento	SUB .0,[.7]++	R0 7 R0 – M(R7), R7 7 R7 + n
Indexado	SUB .0,#H'32[.7]	R0 7 R0 – M(32h + R7)
Relativo a base	SUB .0,RB!.2	R0 7 R0 – M(RB + R2)
Relativo a programa	BR $9	PC 7 PC + 9
Relativo a segmento	SUB .0,CS:H'8!H'3A	R0 7 R0 – M(CS*8 + 3Ah)

Direccionamiento implícito

Corresponde a situaciones en las que el operando o su dirección se encuentra en un registro, y éste está implícitamente especificado en el propio codop.

Direccionamiento inmediato

En este tipo de direccionamiento el operando forma parte de la propia instrucción (Figura 8.2). Según el estándar IEEE 694, este tipo de direccionamiento se denota con el prefijo #.

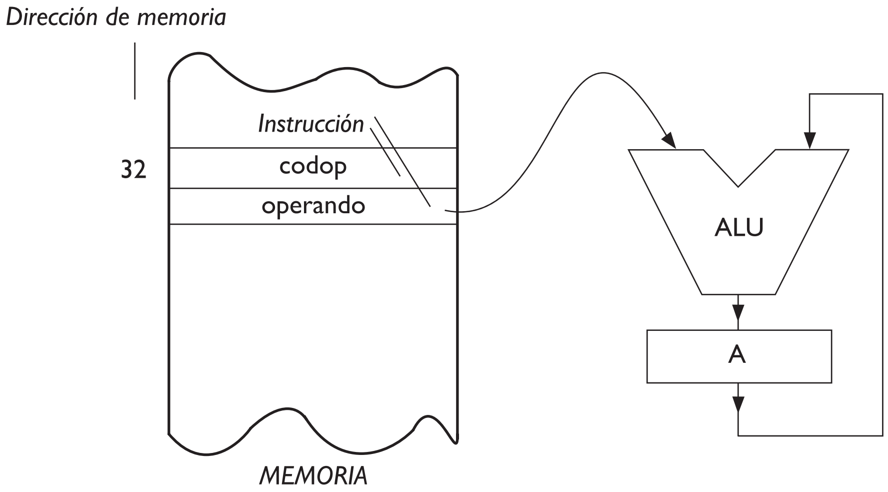


Figura 8.2. *Ejemplo de operación con la ALU con direccionamiento inmediato.*

Direccionamiento directo (o absoluto)

La instrucción contiene la dirección de memoria o del registro donde se encuentra el operando (Figura 8.3). Según el estándar IEEE 694 si el direccionamiento se hace a un registro, el número de orden de éste debe ser precedido por un punto (.), y si corresponde a una posición de memoria el prefijo es una barra inclinada (/) (véase ejemplos en la Tabla 8.1).

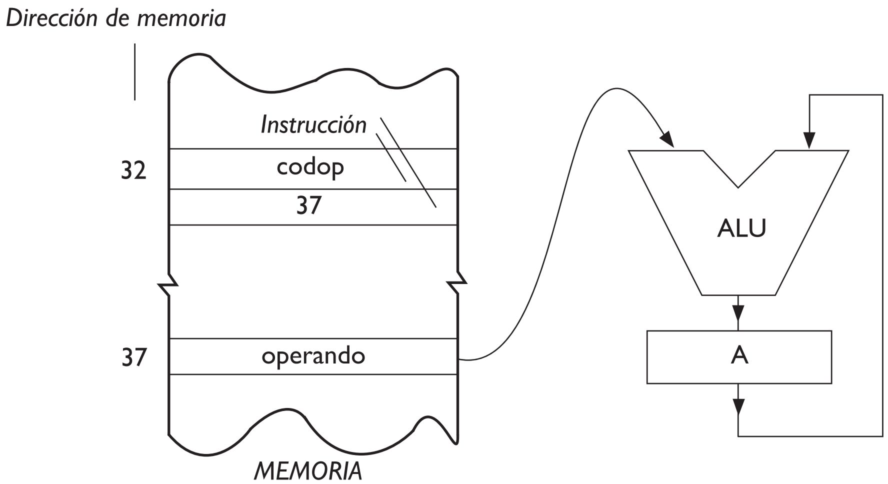


Figura 8.3. *Ejemplo de operación con la ALU con direccionamiento directo a memoria.*

Direccionamiento indirecto

En la instrucción se indica el **puntero** o lugar (posición de memoria o código de registro) donde se encuentra la *dirección* del operando (Figura 8.4).

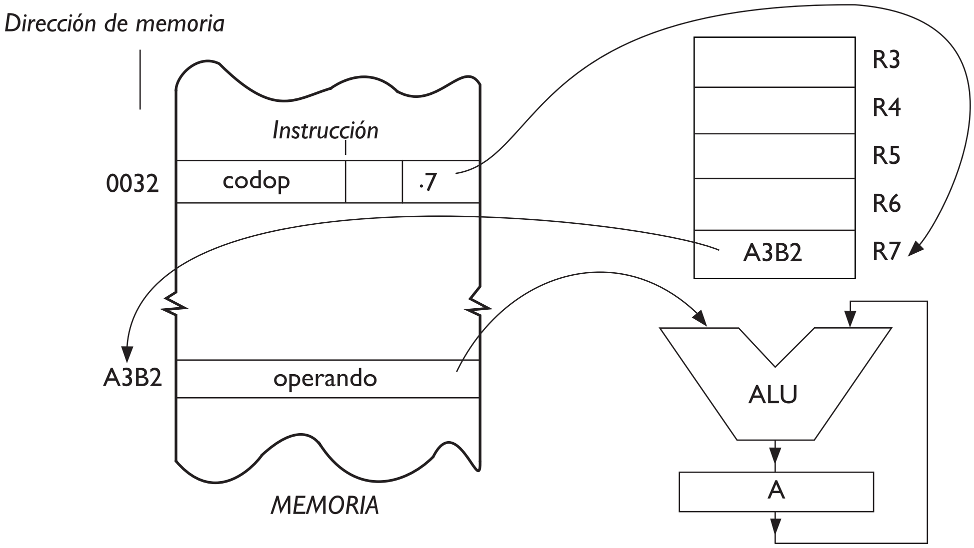


Figura 8.4. *Ejemplo de operación en la ALU utilizando direccionamiento indirecto a memoria a través de registro.*

El direccionamiento indirecto frecuentemente va combinado con un incremento o decremento automático del puntero que indica la dirección de memoria. El incremento, si no se especifica, coincide con el número de palabras que ocupa el operando. Esta actualización del puntero puede realizarse antes o después de la captación del dato; es decir, pueden existir las siguientes combinaciones (*lo* es la longitud del operando).

Autopost-incremento	[.7]++	referencia a *M(R7)* y se hace $R7 \leftarrow R7 + lo$
Autopost-decremento	[.7]−−	referencia a *M(R7)* y se hace $R7 \leftarrow R7 - lo$
Autopreincremento	++[.7]	hace $R7 \leftarrow R7 + lo$ y referencia a *M(R7)*
Autopredecremento	−−[.7]	hace $R7 \leftarrow R7 - lo$ y referencia a *M(R7)*

En la Tabla 8.1 se muestra un ejemplo de este tipo de direccionamiento.

Direccionamiento indexado

Interviene un registro especial o **registro índice** (*X*), que almacena un desplazamiento. La instrucción contiene la dirección de referencia (*DIRR*). La dirección efectiva (*DIRF*) es obtenida por el procesador sumando a la dirección de referencia *(DIRR)* el valor *i* contenido en el registro índice *(X);* es decir:

$$DIRF \leftarrow DIRR + i$$

Este tipo de direccionamiento está proyectado para que el programador pueda, por ejemplo, recorrer con facilidad, sin más que ir incrementando el registro índice dentro de un lazo, los valores de una tabla almacenada en memoria. El programador dispone de instrucciones máquina para utilizar los distintos registros índices que pudiese haber. En la Figura 8.5 se incluye un ejemplo.

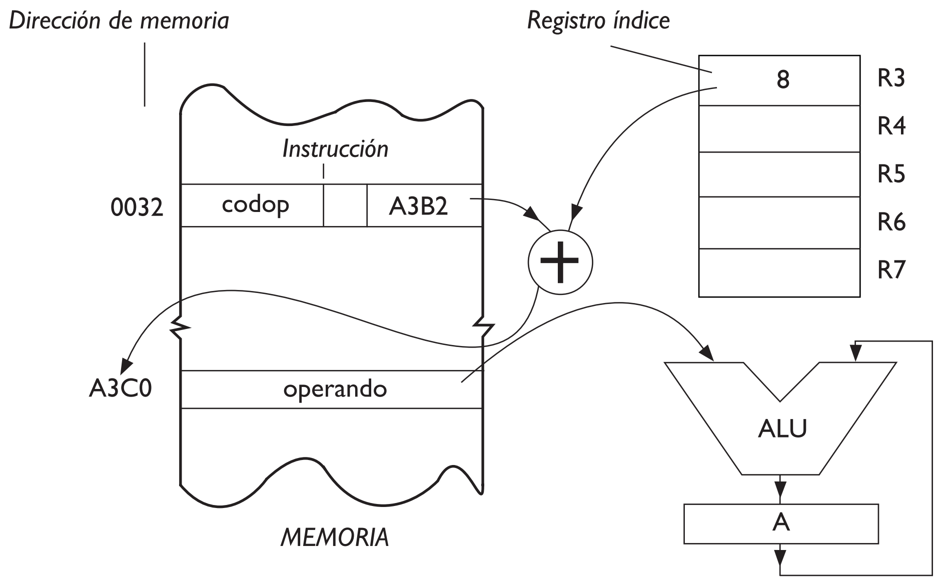


Figura 8.5. *Esquema de obtención de una dirección efectiva con indexación.*

Según el estándar IEEE 694, este tipo de direccionamiento se denota como el direccionamiento indirecto pero anteponiendo la dirección de referencia: *DIRR[X].* En la Tabla 8.1 se muestra un ejemplo de indexación de la dirección de referencia H'32 con el registro índice *R7.* Si se utilizase post-incremento, en la propia instrucción, después de acceder al dato se incrementaría el valor de *R7.*

Direccionamientos relativos

La dirección efectiva se obtiene añadiendo a una **dirección de referencia** un **desplazamiento** (*offset*) que se da en la instrucción como dirección. La adición o composición entre la dirección de referencia y el desplazamiento puede realizarse por suma directa o yuxtaponiendo las dos direcciones (los bits más significativos son los de la dirección de referencia).

Dependiendo de dónde esté ubicada la dirección de referencia se definen los tipos de direccionamiento relativo que se describen a continuación.

Direccionamiento relativo a base

Hace uso de registros especiales denominados **registros base**, que contienen las direcciones de referencia. La dirección efectiva se obtiene sumando la dirección de referencia con el desplazamiento especificado en la instrucción.

Este tipo de direccionamiento parece igual al direccionamiento indexado, pero en realidad difieren por su uso. En efecto, el direccionamiento relativo a base está proyectado para facilitar el acceso a diferentes campos de una estructura de datos, y es de una gran utilidad para los sistema operativos, de forma que el usuario realiza sus programas utilizando un mapa de direcciones que empiece por la dirección 0, y el sistema operativo cargará el programa en una zona libre de memoria a partir de la posición base B (Sección 9.6). Basta con que el sistema operativo cargue en el registro base correspondiente a este programa el valor B para que las direcciones internas del programa sean totalmente válidas. Si es necesario **reubicar** el programa en otra zona de memoria, basta con modificar la dirección base antes de volverlo a cargar.

Según el estándar IEEE 694, este tipo de direccionamiento se denota indicando *registro_base!desplazamiento* (véase ejemplo en la Tabla 8.1). A veces la dirección de base está implícita, con lo que únicamente es necesario indicar *!desplazamiento*.

Direccionamiento relativo a contador de programa

La dirección de referencia es la contenida en el PC. La dirección efectiva se obtiene sumando al contenido del contador de programa (PC) el desplazamiento incluido en la propia instrucción (Figura 8.6). Computadores que utilizan este tipo de direccionamiento para las instrucciones de salto son DEC VAX y MIPS R2000. Los saltos en el 80386 y 80486 permiten varios tipos de direccionamientos, siendo el relativo a contador de programa el más usual.

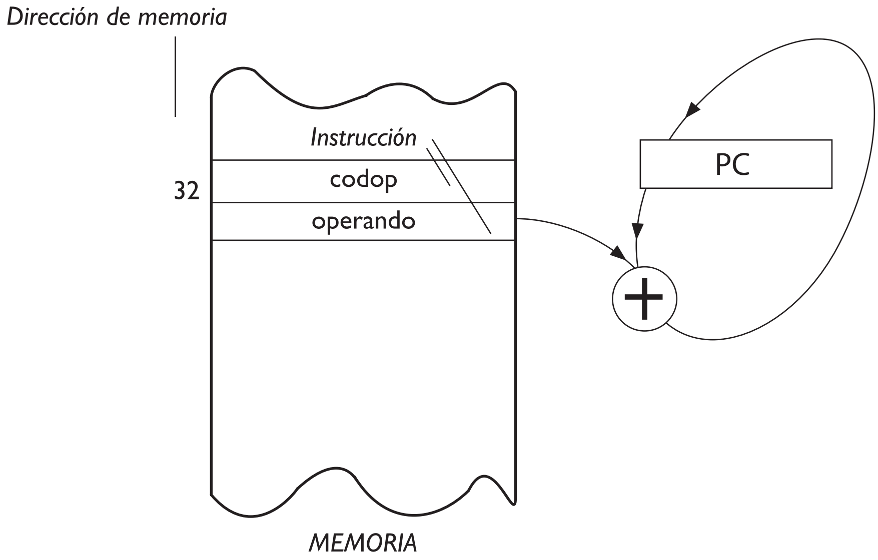


Figura 8.6. *Esquema de cálculo de la dirección efectiva en una instrucción de salto con direccionamiento relativo a programa.*

Este direccionamiento es de interés, por ejemplo, para la realización de bifurcaciones en un programa. El usuario se limita a incluir en la instrucción el número de posiciones de memoria hacia delante o hacia atrás que debe saltar con respecto a la dirección de memoria de la propia instrucción. De esta forma las direcciones de salto especificadas son "relativas" y por tanto independientes de las posiciones reales donde se cargue el programa en la memoria principal.

Según el estándar IEEE 694, este tipo de direccionamiento se denota con el prefijo $. En el ejemplo de la Tabla 8.1, el control saltaría a 9 posiciones de memoria por delante de la dirección que marca el contador de programa después de la fase de captación de la instrucción (BR es la instrucción *saltar*, "branch" en inglés).

Direccionamiento relativo a segmento

El procesador dispone de registros de segmento, por ejemplo, registro de segmento de datos (DS), registro de segmento de código (CS) y registro de segmento de pila (SS) (para pila simulada en memoria principal). La dirección efectiva se obtiene sumando al registro de segmento escalado (4 posiciones, por ejemplo) el desplazamiento especificado en la instrucción. Escalar es desplazar a la izquierda; es decir, multiplicar por 2, 4, 8, 16, etc. (escalar 4 posiciones es

multiplicar por 16). Así el direccionamiento de este tipo en un microprocesador 8086 se realiza según muestra el siguiente ejemplo:

Registro de segmento:	DS	→ A754	(16 bits)
Dirección relativa (desplazamiento)		→ 1235	(16 bits)

Dirección efectiva:	A7540	
	01235	
	A8775	(20 bits)

Según el estándar IEEE 694, el escalonamiento puede denotarse con el subfijo ":", tal como muestra el ejemplo de la Tabla 8.1.

Una cuestión que hay que aclarar es que cada computador (en función de su arquitectura) tiene unos modos de direccionamiento u otros. Un factor importante al comparar dos procesadores es analizar los tipos de direccionamiento que permite, en función de la frecuencia de su utilización por los programas a ejecutar.

8.2 Instrucciones máquina

Según indicamos en la Sección 1.4, las instrucciones se pueden clasificar en 4 grupos:

a) de transferencia de información;
b) aritmético-lógicas y de desplazamientos;
c) transferencias de control (saltos condicionales, bifurcaciones, llamadas a procedimientos y retornos de procedimientos) ; y
d) misceláneas.

Cada instrucción se suele identificar con un nemónico que hace referencia a la función que realiza la instrucción.

Una instrucción máquina contiene diversas informaciones o campos. Además del código de operación, que identifica la instrucción, puede contener un campo de modo de direccionamiento (que puede estar implícito en el codop) y las direcciones de los operandos. Dependiendo del computador, puede ser que en la instrucción se expliciten uno, dos o más operandos, cada uno de los cuales puede direccionarse de forma distinta. En la Tabla 8.2 se indican formas usuales de realizar una instrucción de suma con procesadores de distinto número de operandos.

Tabla 8.2. *Tipos de instrucciones, según el número de operandos.*

Tipo de instrucción	Ejemplo	Explicación
Un operando	ADD /direcc	A ← A + M(direcc)
Dos operandos	ADD .0, .2	R0 ← R0 + R2
Tres operandos	ADD .0, .1, .2	R0 ← R1 + R2
Cuatro operandos	DIV .1, .2, .3, .4	R1 (*cociente*), R2 (*resto*) ← R3/R4

En la Figura 8.7 se muestran algunos formatos de computadores clásicos, y en la Figura 8.8 los del procesador MIPS, que al ser un procesador de tipo RISC tiene unos formatos muy regulares (Sección 12.1).

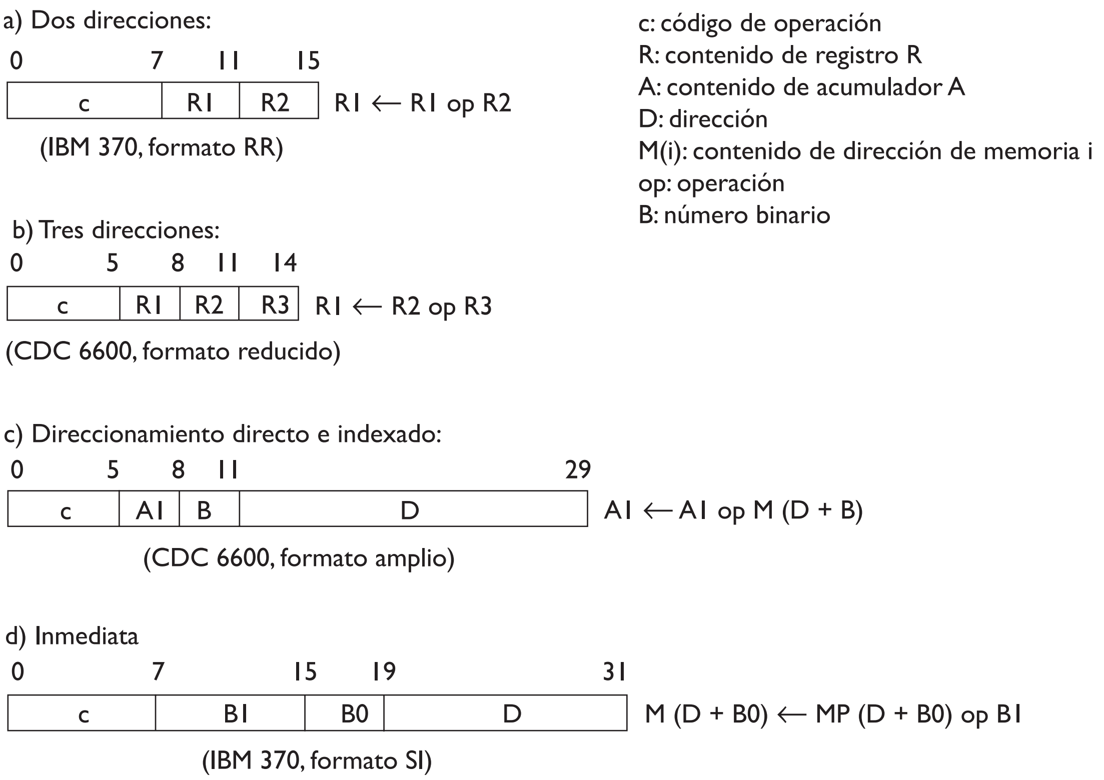


Figura 8.7. *Ejemplos de formatos de instrucciones máquina en algunos computadores clásicos.*

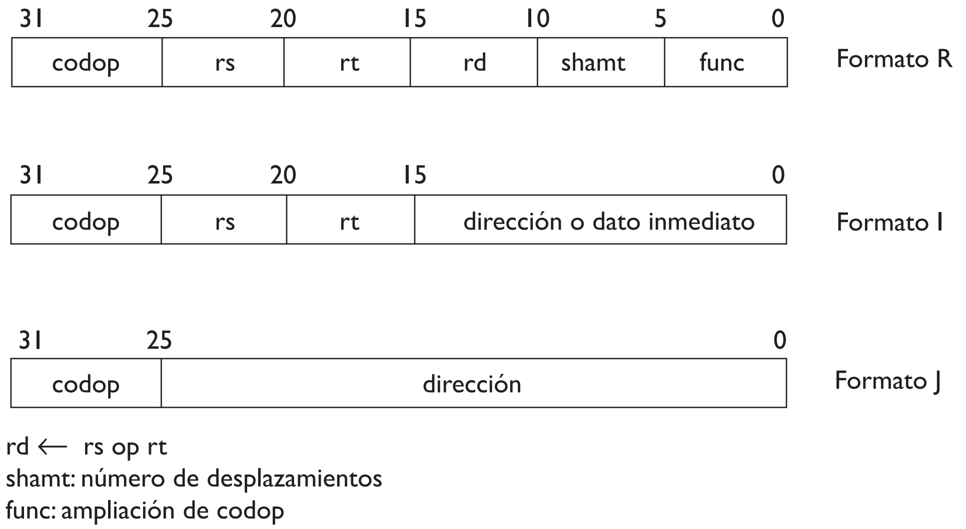


Figura 8.8. *Formatos básicos de las instrucciones de un procesador RISC (procesador MIPS).*

Una instrucción máquina puede ocupar una o varias palabras de memoria. Ejemplos de sistemas en los que una instrucción máquina ocupa más de una palabra lo constituyen los microprocesadores de 8 bits (véase Figura 8.9). En estos existen instrucciones de 1, 2 ó 3 bytes. El primero contiene siempre el código de operación. En las instrucciones de 2 bytes, el segundo, dependiendo del código de operación, puede contener o un dato con el que hay que operar (instrucciones con direccionamiento inmediato) o la dirección de un puerto de E/S. En las de 3 bytes el segundo y tercero contienen un dato inmediato de 16 bits o una dirección de memoria, para localizar un dato o dirección a donde hay que saltar, o un dato con el que hay que operar.

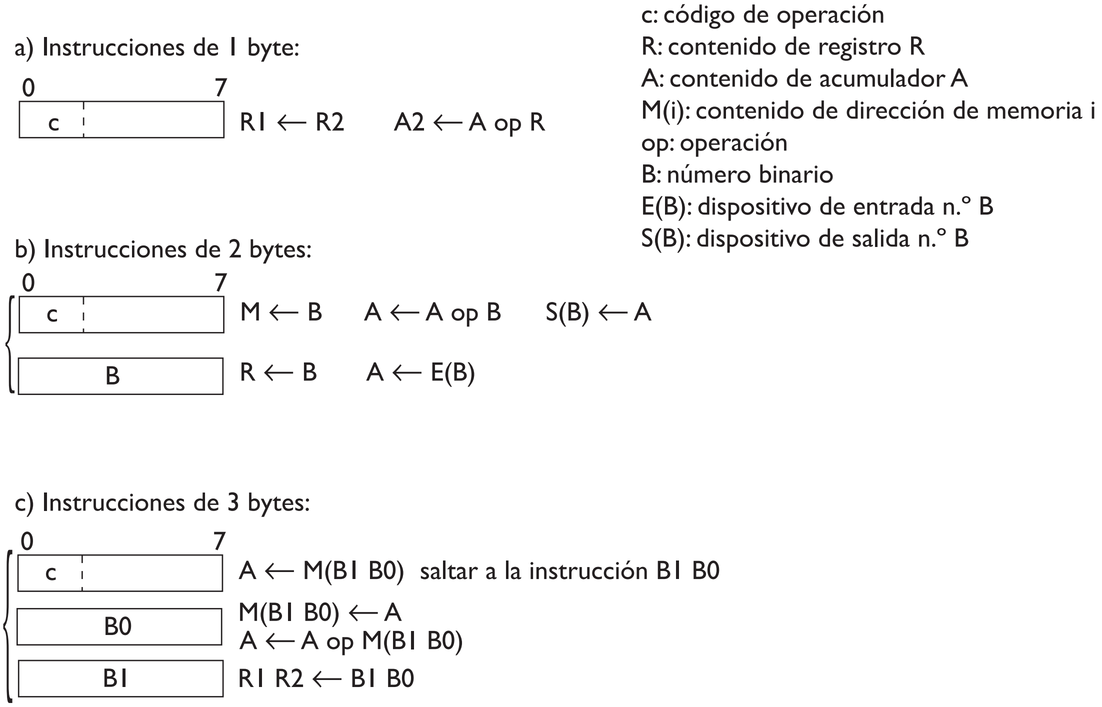


Figura 8.9. *Ejemplos de formatos de instrucciones máquina de un microprocesador (Intel 8085).*

Un ejemplo más notable de instrucciones que ocupan más de una palabra lo constituye el repertorio de instrucciones IA-32, utilizado en los procesadores de Intel, del 8386 al Pentium 4. Hay una gran variedad de tamaños de instrucciones, que van desde uno a doce bytes, siguiendo el esquema general de la Figura 8.10. Todas las instrucciones tienen el campo *codop* (código de operación), pero el que tengan uno u otro de los otros campos depende del tipo de instrucción.

codop	**Dirección**	**Desplazamiento**	**Valor**
1 ó 2 Bytes	1 ó 2 Bytes	1 ó 4 Bytes	1 ó 4 Bytes

Figura 8.10. *Esquema que muestra los distintos posibles tamaños de las instrucciones del repertorio IA-32.*

También existen computadores en los que una misma palabra puede contener varias instrucciones. Así, por ejemplo, en los Cyber 170 serie 800 (Control Data) hay instrucciones de 15 y 30 bits pudiéndose empaquetar en una palabra de 60 bits:

4 instrucciones de 15 bits, o

2 instrucciones de 15 bits y 1 de 30 bits.

Mención aparte merecen los **procesadores VLIW** (*Very Long Instruction Word*, palabra de instrucción muy larga) en los que el compilador genera palabras en las que incluyen varias instrucciones máquina que puede ejecutar el procesador en paralelo. Un ejemplo de este tipo de procesadores es el Itanium 2 (IA-64).

8.3 Descripción de un computador didáctico elemental al nivel de lenguaje máquina

En esta sección se describe la versión 2 de un **Computador Didáctico Elemental (CODE-2)** a nivel de lenguaje máquina.

CODE-2 es un computador de tipo von Neumann que contiene las unidades típicas de este modelo: entradas, salidas, unidad de control, unidad de procesamiento y memoria de datos e instrucciones. Se ha diseñado con los elementos básicos de cualquier tipo de computador, pero simplificando su complejidad para comprenderlo más fácilmente y poder diseñarlo completamente en el Capítulo 9. CODE-2 es de 16 bits de longitud de palabra y tiene tan sólo 16 instrucciones máquina. En el presente capítulo únicamente se va a describir su funcionamiento al *nivel de máquina convencional* (lenguaje máquina y ensamblador, véase Sección 1.3), por lo que fundamentalmente se van a incluir los conceptos y elementos de CODE-2 visibles en este nivel de complejidad.

8.3.1 Elementos a los que se tiene acceso desde el lenguaje máquina

En la Figura 8.11 se muestra un esquema simplificado de CODE-2, que incluye los elementos accesibles directamente desde el lenguaje máquina; éstos son:

- **Banco de registros** (RF, *register file*), compuesto de 16 registros r0,...,rF. El registro rE se utiliza como puntero de pila (SP) y el rD como **registro de dirección**, para almacenar posiciones de memoria con objeto de realizar direccionamientos indirectos e indexados. A pesar de la dedicación específica de los registros rE y rD, el programador puede utilizarlos también para otros cometidos.
- **Unidad aritmético lógica** (ALU), con la que se pueden realizar las operaciones aritméticas de suma y resta (entero en complemento a 2), operación lógica NAND, y desplazamientos lógicos del acumulador a derecha e izquierda y desplazamiento aritmético a la derecha. Estas operaciones se describirán en detalle en la Sección 6.3.3.
- **Biestables indicadores** (FF, *flag flip-flops*). Se activan después de realizar en la ALU las instrucciones de suma, resta, NAND y de desplazamientos; todo ello de acuerdo con el resultado de la operación correspondiente. Se incluyen biestables de cero (Z), signo (S), acarreo (C) y desbordamiento (V). En la Tabla 8.3 se indican las instrucciones que actúan sobre los biestables indicadores; el resto de instrucciones los deja inalterados.
- **Memoria principal** (M), de 2^{16} = 64 K palabras de 16 bits (128 KBytes).

- **Puertos de entrada** (IP, *input port*). El lenguaje máquina admite la inclusión de hasta 256 puertos de entrada (IP00 a IPFF) de periféricos.
- **Puertos de salida** (OP, *output ports*). El lenguaje máquina admite la inclusión de hasta 256 puertos de salida (OP00 a OPFF) de periféricos.

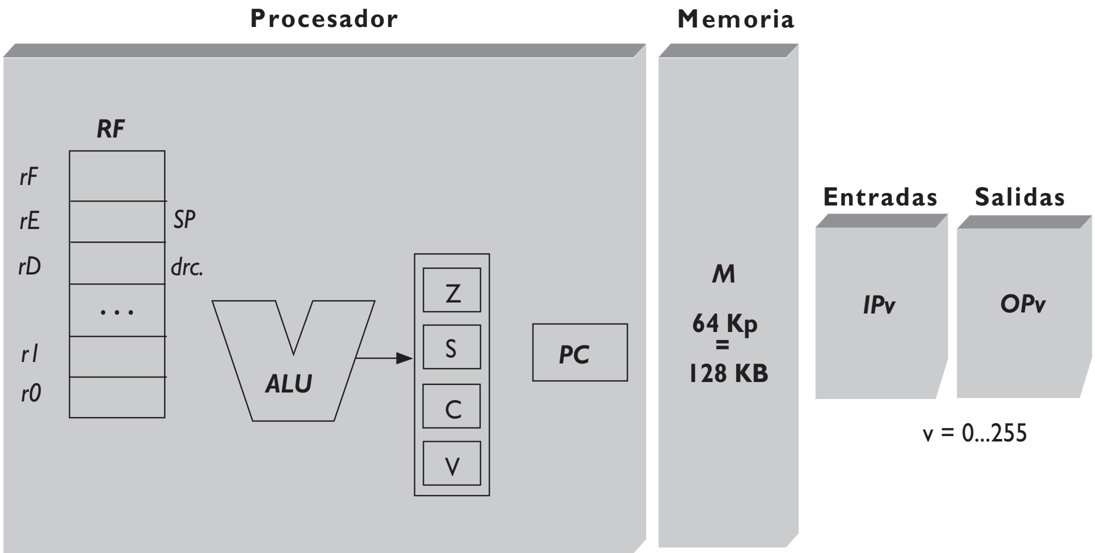


Figura 8.11. *Esquema simplificado del Computador Didáctico Elemental (CODE-2).*

Tabla 8.3. *Valores que toman los biestables indicadores en función del resultado de las instrucciones que se indican.*

Instrucción	Biestable			
	Z	S	C	V
ADDS	*	*	*	*
SUBS	*	*	*	*
NAND	*	*	–	–
SHL	*	*	*	–
SHR	*	0	*	–
SHRA	*	*	*	–

Notas:

1. Se indican sólo las instrucciones que actúan sobre los biestables indicadores.
2. * significa que el biestable se modifica de acuerdo con el resultado de la operación.
3. – significa que el biestable no se modifica.

8.3.2 Formatos de instrucciones y datos

El repertorio de instrucciones de CODE-2 utiliza cinco tipos de formatos (F0 a F4), utilizando cada instrucción uno de ellos (Figura 8.12). Todas las instrucciones (y formatos, por tanto) comienzan con un campo de 4 bits (codop), que corresponde al código de operación. Por tanto CODE-2 puede tener como máximo 16 instrucciones (de hecho sólo tiene 15). Los otros campos de la instrucción dependen del formato que se utilice.

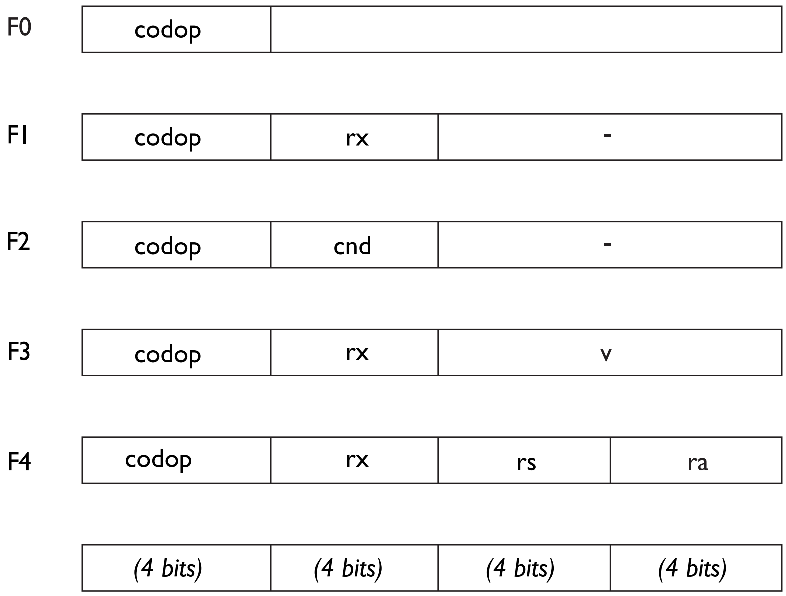


Figura 8.12. *Formatos de las instrucciones de CODE-2.*

- El *formato F0* corresponde a instrucciones que sólo tienen código de operación.
- El *formato F1* dispone del campo de código de operación y un campo de 4 bits para especificar un registro, *rx* (*x = 0, 1, 2,...,E, F),* pudiéndose así referenciar los registros del *r0* a *rF.*
- El *formato F2,* además del campo *codop,* contiene un campo denominado *cnd* que sirve para especificar una condición en las instrucciones de salto o de llamadas a subrutinas, de acuerdo con el significado de la Tabla 8.4.
- El *formato F3,* además del campo codop, contiene otros dos campos: el *rx* (de 4 bits) que sirve para especificar uno de los registros y el *v* utilizado para dar un valor inmediato (dato o dirección) de 8 bits.
- El *formato F4* contiene 4 campos, cada uno de ellos de 4 bits: *codop, rd* (registro del destino del resultado), *rs* (registro de la fuente del dato) *y ra* (registro auxiliar). Los tres últimos campos sirven para especificar cualquier registro y pueden por tanto tomar un valor de 0 a F.

En la Figura 8.13 se incluye un esquema del formato de los datos. Sencillamente corresponde a números enteros en notación de complemento a 2. Como son datos de 16 bits, los números

mayor y menor que se puede representar (al nivel de lenguaje máquina) son, respectivamente (véase Sección 3.6.1):

[8.1]

$$N_{mayor} = 2^{16-1} - 1 = 32.767; \; N_{menor} = -2^{16-1} = -32.768$$

Tabla 8.4. *Tipos de saltos y llamadas a subrutinas, y códigos utilizados para especificar cada uno de ellos.*

Tipo de salto o llamada	–	cnd
Incondicional	R	0000
Condicional, si biestable de cero es 1	Z	0001
Condicional, si biestable de signo negativo es 1	S	0010
Condicional, si biestable de acarreo es 1	C	0011
Condicional, si biestable de desbordamiento es 1	V	0100

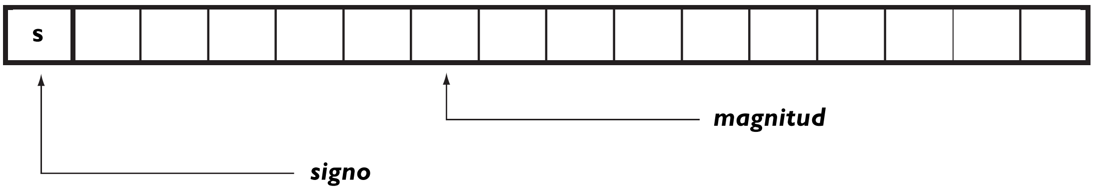


Figura 8.13. *Formato de los datos de CODE-2.*

8.3.3 Repertorio de instrucciones máquina

En la Tabla 8.5 se dan las 16 instrucciones de las que consta el lenguaje máquina de CODE-2. Al escribir $d \leftarrow s$ se indica que el contenido del elemento *s* (fuente) se copia el elemento *d* (destino). El contenido de la fuente en ningún caso cambia; es decir, se "replica" el valor de la fuente en el destino.

Los elementos destino, dependiendo de la instrucción concreta, pueden ser:

- Registros, r0 a rF.
- Una posición de memoria; así, M(x) especifica el contenido de la posición de memoria x.
- El contador de programa, PC.
- Un puerto de salida; así, OPv especifica el contenido del puerto v de salida.

Por otra parte los elementos fuente pueden ser:

- Registros, r0 a rF.
- Una posición de memoria, como en el caso de fuente.
- Un puerto de entrada; así, IPv especifica el contenido del puerto v de entrada.

Los bits particulares de un registro se especifican dándolos entre paréntesis, y se numeran de derecha a izquierda, empezando por 0 (Figura 8.14). Así, *rx(0)* significa el bit menos significativo del registro *rx; rx(15)* representa el bit más significativo del registro *rx,* y *rx(7:0),* representan los 8 bits (byte) menos significativo de *rx.*

Tabla 8.5. *Repertorio de instrucciones de CODE-2.*

Codop		Nombre	Nemónico	Parámetros	Formato	Explicación	Nº ciclos
Binario	Hex						
0000	0	Cargar	LD	rx,[v]	F3	rx ← M(rD + v)	9
0001	1	Almacenar	ST	[v],rx	F3	M(rD + v) ← rx	9
0010	2	Carga inmediata baja	LLI	rx,v	F3	rx(15:8) ← H'00; rx(7:0) ← v	6
0011	3	Carga inmediata alta	LHI	rx,v	F3	rx(15:8) ← v	8
0100	4	Entrada	IN	rx,IPv	F3	rx ← IPv	8
0101	5	Salida	OUT	OPv,rx	F3	OPv ← rx	8
0110	6	Suma	ADDS	rx,rs,ra	F4	rx ← rs + ra	7
0111	7	Resta	SUBS	rx,rs,ra	F4	rx ← rs – ra	7
1000	8	NAND	NAND	rx,rs,ra	F4	rx ← (rs · ra)'	7
1001	9	Desplaza izquierda	SHL	rx	F1	C ← rx(15), rx(i) ← rx(i – 1), i = 15,…,1; rx(0) ← 0	6
1010	A	Desplaza derecha	SHR	rx	F1	C ← rx(0), rx(i) ← rx(i + 1), i = 0,…,14; rx(15) ← 0	6
1011	B	Desplaza arit. dcha.	SHRA	rx	F1	C ← rx(0), rx(i) ← rx(I + 1), i = 0,…,14	6
1100	C	Salto	B-	cnd	F2	Si cnd se cumple, PC ← rD	6
1101	D	Subrutina	CALL-	cnd	F2	Si cnd se cumple, rE ← rE – 1, M(rE) ← PC, PC ← rD	6/10
1110	E	Retorno	RET	-	F0	PC ← M(rE); rE ← rE + 1	8
1111	F	Parar	HALT	-	F0	Parar	6

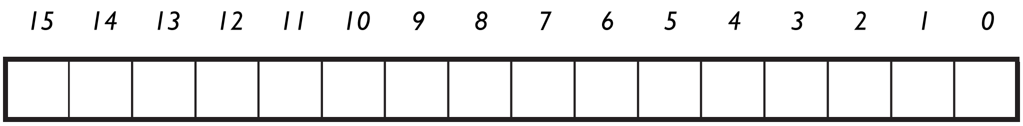


Figura 8.14. *Identificación de los bits individuales de una palabra.*

Atendiendo al fin didáctico de CODE-2, su repertorio de instrucciones se ha escogido suficientemente limitado como para que el excesivo número de instrucciones no sea una dificultad para aprender a utilizarlo, y lo suficientemente amplio como para poder escribir una gran variedad de programas.

A cada instrucción se le ha asignado un nemónico (columna 4 de la Tabla 8.5) que resulta muy cómodo de utilizar cuando se está redactando el programa, como paso previo a la utilización de los códigos binarios que son los que forman el programa en lenguaje máquina (los nemónicos se usarán también en el lenguaje ensamblador, Sección 8.4).

Por otra parte cada instrucción, además del nemónico del código de operación, puede tener ningún, uno, dos o tres parámetros, dependiendo de que su formato sea de tipo F0, F1, F2/F3 o F4, respectivamente, tal como se indica en la quinta columna de la Tabla 8.5.

En la última columna de la Tabla 8.5 se incluye el número de ciclos de reloj que consume cada instrucción. Consultando esta columna, y conociendo la frecuencia de reloj, se puede obtener el tiempo que tarda en ejecutarse cada instrucción, o la duración de un programa concreto sumando lo que tarda cada una de sus instrucciones. Llamando N_{Ci} al número de ciclos de la instrucción *i,* y *F* a la frecuencia de reloj de CODE-2, dicha instrucción tarda en ejecutarse:

[8.2]

$$t_i = N_{Ci} \cdot T = \frac{N_{Ci}}{F}$$

donde *T* es el periodo del reloj **(tiempo de ciclo)**.

A continuación vamos a describir cada una de las instrucciones, siguiendo el orden de la Tabla 8.5.

LD (cargar un registro con un dato de la memoria)

Instrucción: cargar un registro con un dato de la memoria.
Código de operación: 0000 (0 en hexadecimal).
Tipo de formato: F3 (Figura 8.12).
Instrucción en nemónico: LD rx,[rD + v] o LD rx,[v].
Descripción de función a realizar: rx ← M(rD + v).

Esta instrucción carga en un registro el contenido de la posición de memoria cuya dirección se obtiene sumando el contenido del registro *rD* con el valor del campo v de la propia instrucción.

Obsérvese que la forma de direccionar indicada permite implementar cuatro tipos de direccionamiento (Sección 8.1), que se resumen en la Tabla 8.6:

- Direccionamiento directo, si *rD* es *H'0000*.
- Direccionamiento indirecto a través de registro, si el campo v de la instrucción es *H'00.*
- Direccionamiento indexado, actuando *rD* como registro índice.
- Direccionamiento relativo a base, actuando *rD* como registro base (dirección de referencia) y siendo *v* el desplazamiento, que se incluye en la propia instrucción.

Tabla 8.6. *Direccionamientos implementables a partir de la suma del contenido de un registro y un campo de la instrucción.*

Tipo de direccionamiento	Función o contenido del registro rD	Función o contenido del campo v	Dirección efectiva
Directo	H'0000	dirección	v
Indirecto a través de registro	dirección	H'00	rD
Indexado	Registro índice (i)	Dirección de referencia (DIRR)	v + rD
Relativo a base	Registro base	desplazamiento	rD + v

EJEMPLO 8.1

Supóngase que en el registro rD se encuentra el valor H'003B y se ejecuta la instrucción LD r1, [A7], que en código máquina es 01A7 = B' 0000 0001 1010 0111, ubicada en la posición 00AB de memoria.

En la Figura 8.15 puede verse un esquema de las operaciones realizadas para la ejecución de esta instrucción. Los valores indicados en los registros y en la memoria son los que resultan después de ejecutar la instrucción. Como el codop es H'0, se trata de la instrucción LD, cuyo formato es F3, con lo que el registro (rx) referenciado en la instrucción es el r1 y el campo v es H'A7. Por consiguiente, al ejecutarse esta instrucción se llevará al registro r1 el valor almacenado en la posición de memoria rD + v = 003B + A7 = 00E2. Obsérvese que el contador de programa contiene la dirección de la siguiente instrucción a ejecutar, en este caso 00AC.

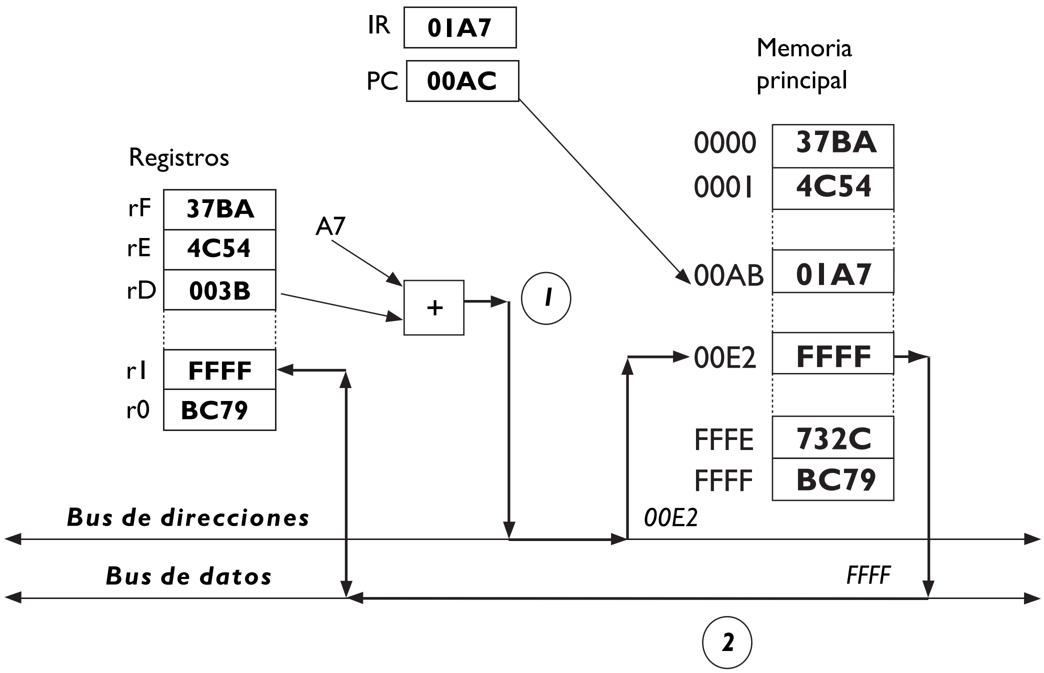


Figura 8.15. *Ejemplo de ejecución de una instrucción de carga de un registro con un valor de memoria (LD r1, [A7]).*

ST (almacenar el contenido de un registro en la memoria)

Instrucción: almacenar el contenido de un registro en la memoria.
Código de operación: 0001 (1 en hexadecimal).
Tipo de formato: F3 (Figura 8.12).
Instrucción en nemónico: ST [rD + v],rx o ST [v],rx.
Descripción de función a realizar: M(rD + v) ← rx.

Esta instrucción carga el contenido del registro *rx*, especificado en la instrucción, en la posición de memoria cuya dirección se obtiene sumando el contenido del registro *rD* con el valor del campo *v* de la propia instrucción.

EJEMPLO 8.2

Supóngase que en la posición de memoria 0019 se encuentra la instrucción H'1000 y que en el registro r0 se encuentra el valor H'BC79, ¿cómo se ejecuta esta instrucción?.

En la Figura 8.16 puede verse un esquema de las operaciones realizadas para la ejecución de esta instrucción. Después de captar la instrucción, IR la contendrá y PC tendrá almacenado la dirección de la siguiente instrucción a ejecutar (001A). Como el codop es H'1 = B'0001, se trata de la instrucción ST, cuyo formato es F3, con lo que el registro rx referenciado en la instrucción es el r0 y el campo v es H'00 Por consiguiente, al ejecutarse esta instrucción se almacenará el contenido del registro r0, es decir el valor BC79, en la posición de memoria rD + v = 003B + 00 = 003B.

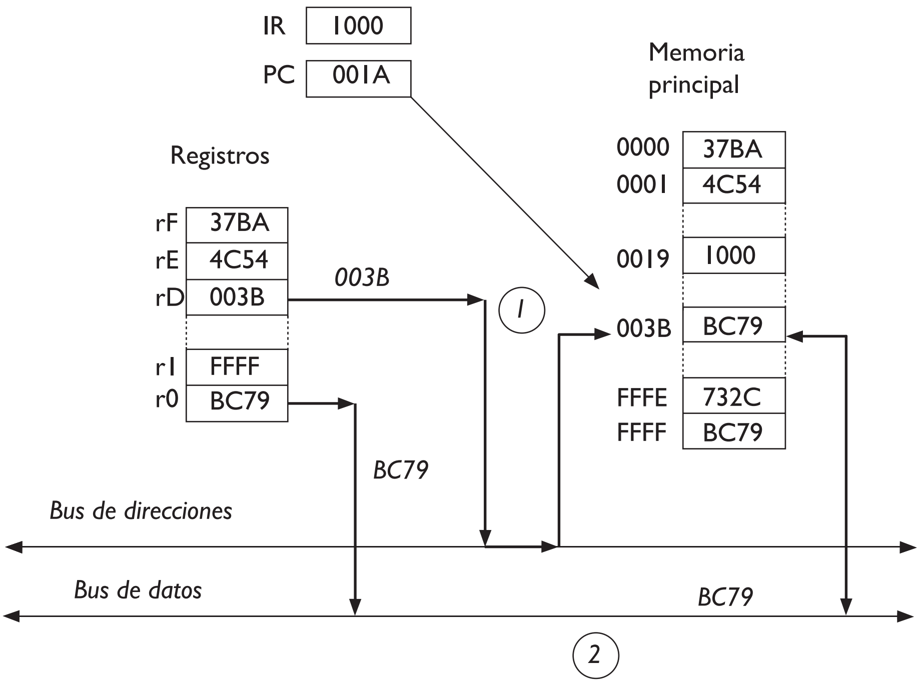


Figura 8.16. *Ejemplo de ejecución de una instrucción de memorización (ST [00], 0).*

LLI (carga inmediata baja)

Instrucción: carga inmediata del byte menos significativo de un registro.
Código de operación: 0010 (2 en hexadecimal).
Tipo de formato: F3 (Figura 8.12).
Instrucción en nemónico: LLI rx,v.
Descripción de función a realizar: rx(15:8) ← H'00; rx(7:0) ← v.

Esta instrucción carga el contenido del byte menos significativo del registro *rx*, especificado en la instrucción, con el valor del campo v de la propia instrucción. El byte menos significativo se pone a *H'00*.

EJEMPLO 8.3

Supóngase que en la posición de memoria 003C se encuentra la instrucción $2F01_H$ y que en el registro r0 se encuentra el valor BC79. ¿Cómo se ejecuta esta instrucción?.

En la Figura 8.17 puede verse un esquema de las operaciones realizadas para la ejecución de esta instrucción. Después de captar la instrucción, IR contendrá la instrucción (2F01) y PC la dirección de la siguiente instrucción a ejecutar (003D). Como el codop es H'2 = B'0010 se trata de la instrucción LLI, cuyo formato es F3, con lo que el registro rx referenciado en la instrucción es el rF y el campo v es H'01. Por consiguiente, al ejecutarse esta instrucción se almacenará en el registro rF el valor 1.

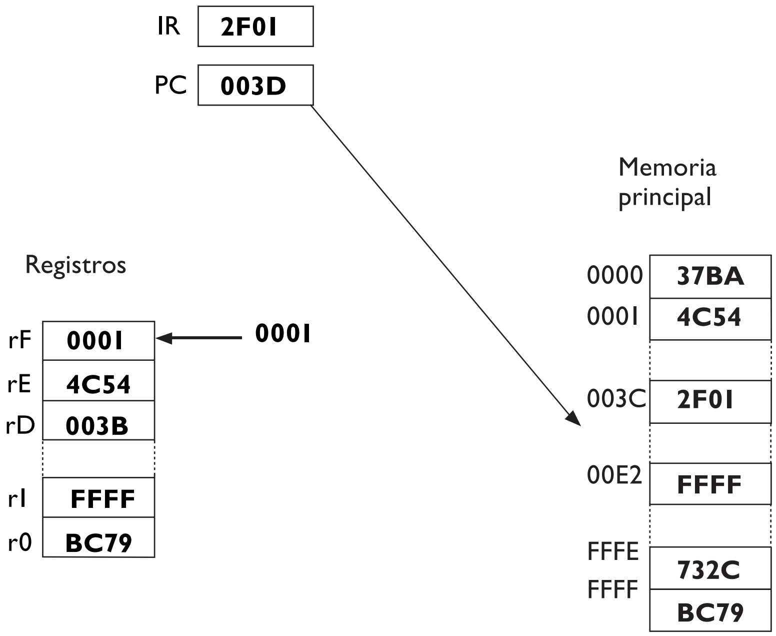


Figura 8.17. *Ejemplo de ejecución de una instrucción de carga inmediata (byte menos significativo) (LLI rF, 01).*

LHI (carga inmediata alta)

Instrucción:	carga inmediata del byte más significativo de un registro.
Código de operación:	0011 (3 en hexadecimal).
Tipo de formato:	F3 (Figura 8.12).
Instrucción en nemónico:	LHI rx,v.
Descripción de función a realizar:	rx(15:8) ← v; el byte menos significativo no se modifica (conserva su valor previo).

Esta instrucción carga el byte más significativo del registro rx, especificado en la instrucción, con el valor del campo v de la propia instrucción. El byte menos significativo conserva su valor previo.

EJEMPLO 8.4

Hacer el byte más significativo de rF igual a 7F.

Esto se puede realizar con la instrucción LHI rF,7F que en código máquina es 3F7F, y que suponemos se ubica en la posición de memoria 003D. En la Figura 8.18 puede verse un esquema de las operaciones realizadas para la ejecución de esta instrucción. Después de captar la instrucción, el registro IR contendrá la instrucción (3F7F), y PC la dirección de la siguiente instrucción a ejecutar (003E). Como el codop es H'3 = B'0011 se trata de la instrucción LHI, cuyo formato es F3, con lo que el registro rx referenciado en la instrucción es el rF y el campo v es H'7F. Por consiguiente, al ejecutarse esta instrucción se almacenará en el byte más significativo del registro rF el valor 7F. Si previamente en él estuviese 0001, el nuevo contenido de este registro será: 7F01.

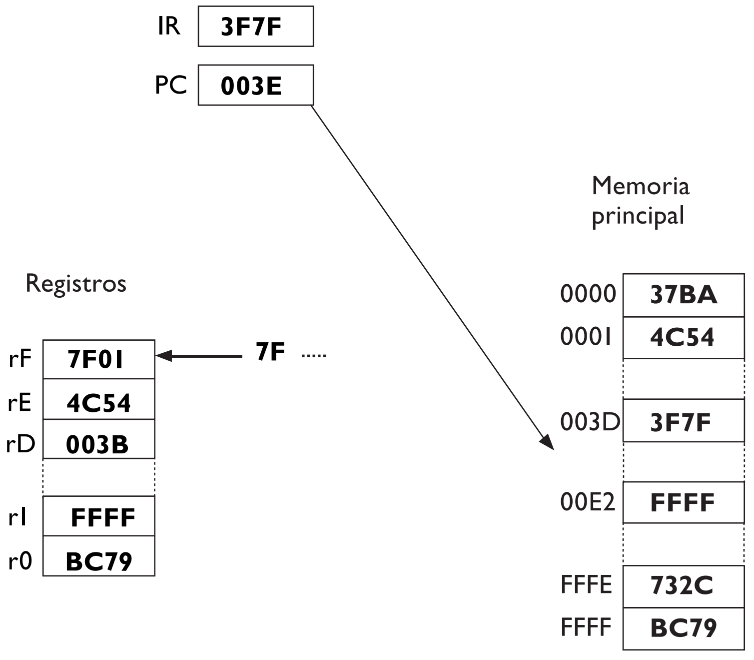


Figura 8.18. *Ejemplo de ejecución de una instrucción de carga inmediata (byte más significativo) (LHI rF, 7F).*

IN (entrada)

Instrucción: captar el contenido de un puerto de entrada.
Código de operación: 0100 (4 en hexadecimal).
Tipo de formato: F3 (Figura 8.12).
Instrucción en nemónico: IN rx,IPv.
Descripción de función a realizar: rx $\leftarrow$ IPv.

Esta instrucción capta el contenido del puerto *IPv* (donde v se especifica en la propia instrucción) y lo lleva al registro *rx*, dado también en la propia instrucción.

EJEMPLO 8.5

Supóngase que se desea, en la posición 0023 de memoria, llevar el contenido del puerto de entrada IP2 al registro r1. ¿Con que instrucción se podría realizar esta acción?.

La instrucción será IN r1,IP2 o, en código máquina, 4102. En la Figura 8.19 puede verse un esquema de las operaciones realizadas para la ejecución de esta instrucción. Después de captar la instrucción, el registro IR contendrá la instrucción (4102), y PC la dirección de la siguiente instrucción a ejecutar (0024). Como el codop es H'4 = B'0100 se trata de la instrucción IN, cuyo formato es F3, con lo que el registro rx referenciado en la instrucción es el r1 y el campo v es H'02 que en este caso hace referencia al puerto de entrada IP2. Por consiguiente, al ejecutarse esta instrucción se captará el valor del puerto IP2 (3BCF, en la figura) y se almacenará en el registro r1.

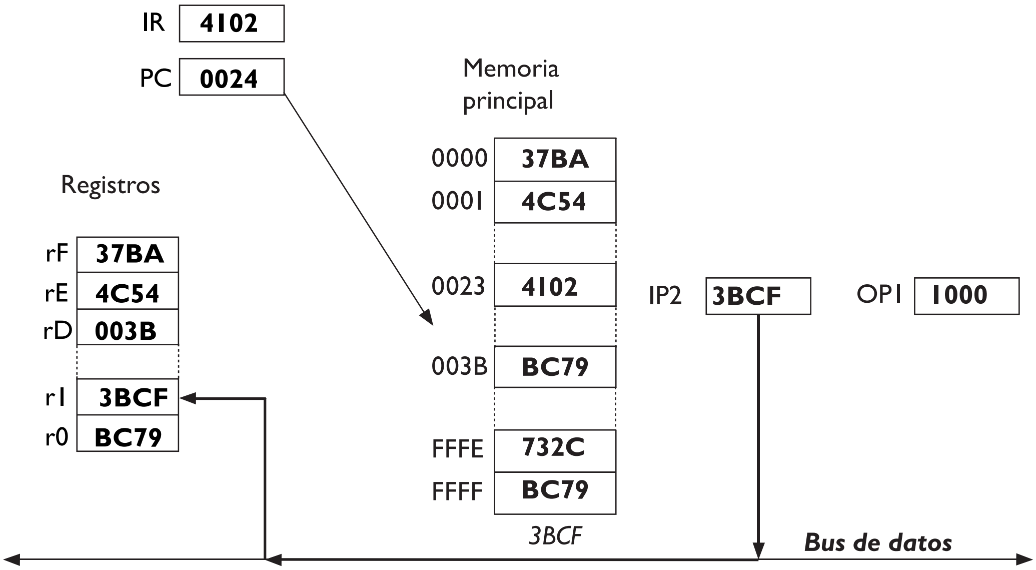


Figura 8.19. *Ejemplo de ejecución de una instrucción de entrada (IN r1,IP2).*

Como se verá más adelante (Sección 8.3.6) en la implementación realizada de CODE-2 para la realización de ejercicios y prácticas, las entradas se dan a través de un teclado hexadecimal cuya información de graba en el puerto de entrada *1* (*IP1*). Es decir, siempre que se ejecute la instrucción *IN rx,IP01*, se llevará la última información introducida por el teclado al registro *rx*.

OUT (salida)

Instrucción: llevar el contenido de un registro a un puerto de salida.
Código de operación: 0101 (5 en hexadecimal).
Tipo de formato: F3 (Figura 8.12).
Instrucción en nemónico: OUT OPv,rx.
Descripción de función a realizar: OPv ← rx.

Esta instrucción lleva el contenido del registro *rx* al puerto de salida *OPv* (donde v se especifica en la propia instrucción).

EJEMPLO 8.6

Supóngase que se desea, en la posición 0024 de memoria, incluir una instrucción para llevar el contenido del registro rD al puerto de salida OP1. ¿Qué instrucción debe utilizarse?

La instrucción será OUT OP1, rD, o, en código máquina, 5D01. En la Figura 8.20 puede verse un esquema de las operaciones realizadas para la ejecución de esta instrucción. Después de captar la instrucción, el registro IR contendrá la instrucción (5D01), y PC la dirección de la siguiente instrucción a ejecutar (0025). Como el codop es H'0101 = B'0101, se trata de la instrucción OUT, cuyo formato es de tipo F3, con lo que el registro rx referenciado en la instrucción es el rD y el campo v es H'01, que en este caso hace referencia al puerto de salida OP1. Por consiguiente, al ejecutarse esta instrucción se llevará el valor del registro rD (003B, en la figura) al puerto de salida OP1.

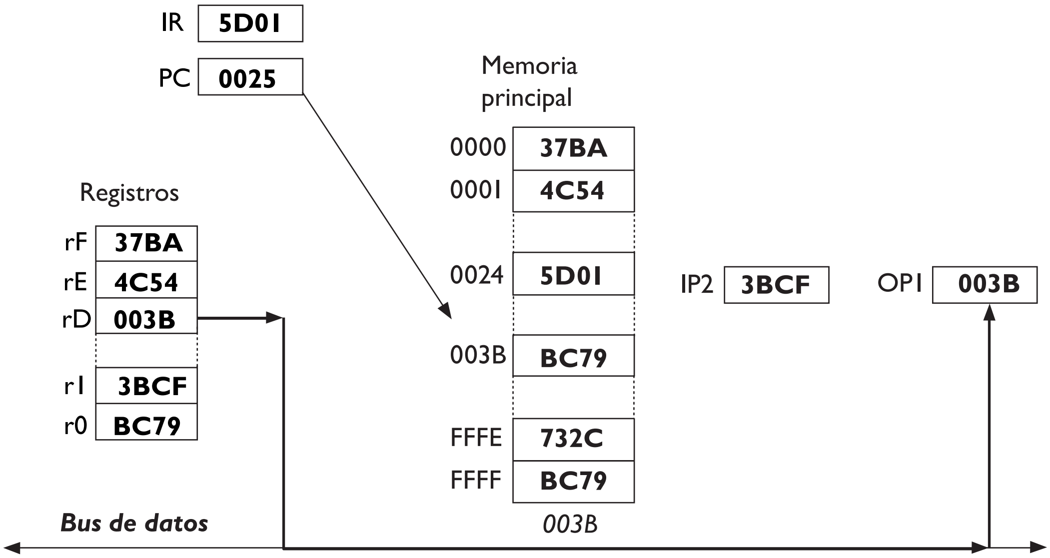


Figura 8.20. *Ejemplo de ejecución de una instrucción de salida (OUT OP1,rD).*

Como se verá más adelante (Sección 8.3.6) en la implementación realizada de CODE-2 para la realización de ejercicios y prácticas, las salidas se dan a través de dos puertos de salida (*OP1* y *OP2*), cada uno de ellos está conectado a 4 indicadores de 7 segmentos, que muestran el valor hexadecimal del puerto respectivo. Así, la instrucción *OUT OP1,rx* hace que se muestre en el puerto de salida 1, el contenido, en hexadecimal, del registro *rx*. Por otra parte, la instrucción *OUT OP2,rx* visualiza el contenido del mismo registro, pero en el puerto *OP2*.

ADDS (suma)

Instrucción: suma del contenido de 2 registros.
Código de operación: 0110 (6 en hexadecimal).
Tipo de formato: F4 (Figura 8.12).
Instrucción en nemónico: ADDS rx,rs,ra.
Descripción de función a realizar: rx ← rs + ra.

Esta instrucción suma el contenido de los registros rs y ra, depositando el resultado en el registro rx. La suma se hace para números enteros representados en complemento a 2 (Sección 4.xx y 9.xx). Los tres registros son especificados en tres campos de la propia instrucción. Los cuatro biestables indicadores (*Z, S, C* y *V*) se activan de acuerdo con el resultado de la operación.

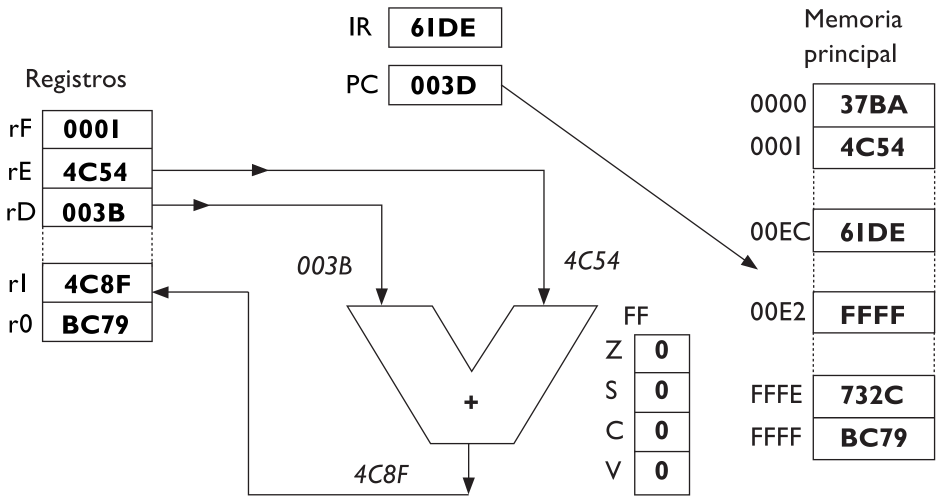


Figura 8.21. *Ejemplo de ejecución de una instrucción de suma (ADDS r1,rD,rE).*

EJEMPLO 8.7

¿Qué instrucción habría que usar para sumar el contenido de los registros rD y rE y llevar el resultado al registro r1?

La instrucción será ADDS r1,rD,rE o, en código máquina, 61DE.

En la Figura 8.21 puede verse un esquema de las operaciones realizadas para la ejecución de esta instrucción. Después de captar la instrucción, el registro IR contendrá la instrucción (61DE), y PC la dirección de la siguiente instrucción a ejecutar, suponiendo que la instrucción estuviese en la dirección 003C, el valor nuevo de PC será 003D. Como el codop es H'6 = B'0110, se trata de la instrucción suma, cuyo formato es de tipo F4, con lo que los registros referenciados en la instrucción son: rx = r1, rs = rD y ra = rE. Teniendo en cuenta los contenidos de los registros indicados en la figura, el resultado de la operación (r1 ← rD + rE) será:

rD →	003B		0000 0000 0011 1011	
rE →	4C54	+	0100 1100 0101 0100	
r1 →			0100 1100 1000 1111	→ 4C8F

Es decir, el nuevo valor de r1 será: 4C8F, y los biestables indicadores: Z = 0, S = 0, C = 0, V = 0.

SUBS (resta)

Instrucción: resta del contenido de 2 registros.

Código de operación: 0111 (7 en hexadecimal).

Tipo de formato: F4 (Figura 8.12).

Instrucción en nemónico: SUBS rx,rs,ra.
Descripción de función a realizar: rx ← rs – ra.

Esta instrucción resta el contenido de los registros *rs* y *ra*, depositando el resultado en el registro rx. La resta se hace para números enteros representados en complemento a 2 (Sección 4.xx y 9.xx). Los tres registros son especificados en tres campos de la propia instrucción. Los 4 biestables indicadores (*Z, S, C* y *V*) se activan de acuerdo con el resultado de la operación.

EJEMPLO 8.8

¿Qué instrucción se puede usar para restar el contenido de los registros rD y rE y llevar el resultado al registro r0?

La instrucción será SUBS r0,rD,rE o, en código máquina, 70DE.

En la Figura 8.22 puede verse un esquema de las operaciones realizadas para la ejecución de esta instrucción. Después de captar la instrucción, el registro IR contendrá la instrucción (70DE), y PC la dirección de la siguiente instrucción a ejecutar, suponiendo que la instrucción estuviese en la dirección 003D, el nuevo valor de PC será 003E. Como el codop es H'7 = B'0111, se tiene la instrucción de resta, cuyo formato es de tipo F4, con lo que los registros referenciados en la instrucción son: rx = r0, rs = rD y ra = rE. Teniendo en cuenta los contenidos de los registros indicados en la figura, el resultado de la operación (r0 ← rD – rE) será:

rD →	E03B		1110 0000 0011 1011	
rE →	4C54	–	0100 1100 0101 0100	
r0 →			1001 0011 1110 0111	→ 93E7

Es decir, el nuevo valor de r1 será: 93E7, y los biestables indicadores: Z = 0, S = 1, C = 0, V = 0.

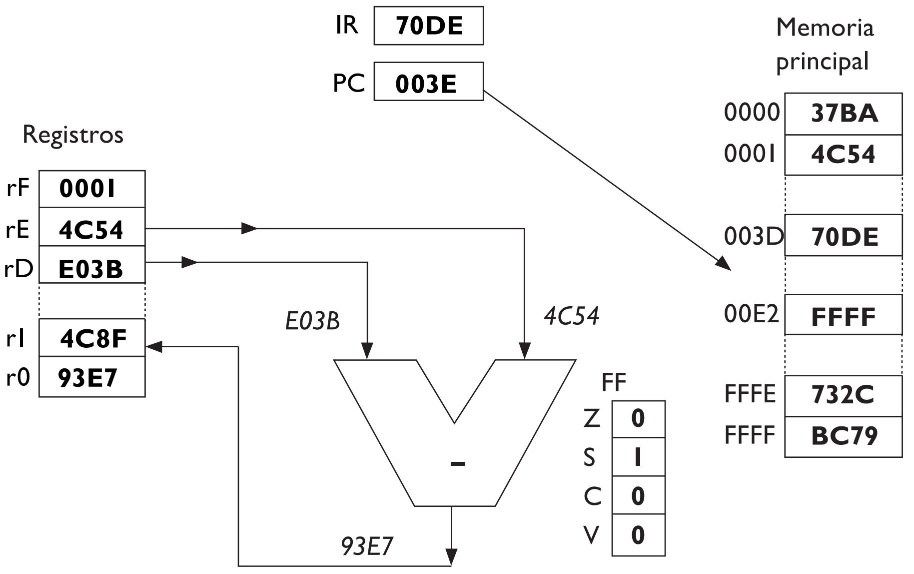


Figura 8.22. *Ejemplo de ejecución de una instrucción de resta (SUBS r0,rD,rE).*

NAND (operación lógica NAND)

Instrucción: operación lógica NAND bit a bit del contenido de 2 registros.
Código de operación: 1000 (8 en hexadecimal).
Tipo de formato: F4 (Figura 8.12).
Instrucción en nemónico: NAND rx,rs,ra.
Descripción de función a realizar: rx ← (rs · ra)'.

Esta instrucción efectúa la operación lógica *NAND* bit a bit (es decir, entre bits del mismo orden) de los datos contenidos en los registros *rs* y *ra,* depositando el resultado en el registro *rx*. Los tres registros son especificados en tres campos de la propia instrucción. Se activan los biestables indicadores *Z* y *S* de acuerdo con el resultado de la operación; mientras que *C* y *V* permanecen inalterados. Recuérdese (Sección 5.4.1) que la operación lógica NAND se define como se indica en la Tabla 8.7.

Tabla 8.7. *Operación lógica NAND: c = (a • b).*

a	b	c = (a · b)'
0	0	1
0	1	1
1	0	1
1	1	0

EJEMPLO 8.9

Supóngase que se desea realizar la operación lógica NAND entre los bits de los registros rD y rE y llevar el resultado al registro r1. ¿Qué instrucción se utilizaría?

La instrucción será NAND r1,rD,rE o, en código máquina, 81DE.

En la Figura 8.23 puede verse un esquema de las operaciones realizadas para la ejecución de esta instrucción. Después de captar la instrucción, el registro IR contendrá la instrucción (81DE), y PC la dirección de la siguiente instrucción a ejecutar, suponiendo que la instrucción estuviese en la dirección 003E, el nuevo valor de PC será 003F. Como el codop es H'8 = B'1000, se tiene la instrucción NAND, cuyo formato es de tipo F4, con lo que los registros referenciados en la instrucción son: rx = r1, rs = rD y ra = rE. Teniendo en cuenta los contenidos de los registros indicados en la figura, el resultado de la operación r1 ← (rD · rE)', de acuerdo con la Tabla 8.7, será:

rD →	E03B		1110 0000 0011 1011	
rE →	4C54	NAND	0100 1100 0101 0100	
r0 →			1011 1111 1110 1111	→ BFEF

Es decir, el nuevo valor de r1 será: BFEF, y los biestables indicadores: Z = 0 y S = 1; C y V conservarán sus valores previos.

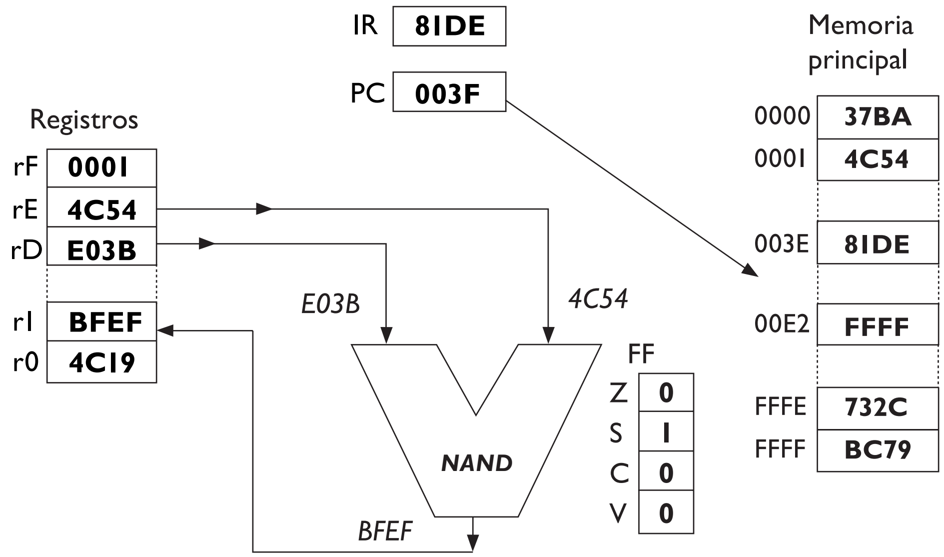


Figura 8.23. *Ejemplo de ejecución de una instrucción NAND (NAND r1,rD,rE).*

SHL (desplazamiento a izquierda)

Instrucción:	desplazamiento a izquierda de los bits de un registro.
Código de operación:	1001 (9 en hexadecimal).
Tipo de formato:	F1 (Figura 8.12).
Instrucción en nemónico:	SHL rx.
Descripción de función a realizar:	C ← rx(15), rx(i) ← rx(i – 1), i = 15,…,1; rx(0) ← 0.

Esta instrucción, Figura 8.24a, desplaza los bits del registro *rx*, especificado en la propia instrucción, una posición a la izquierda. El bit más significativo pasa al biestable de acarreo (*C*) y el bit menos significativo se hace *0*. El biestable de signo (*S*) pasa a tomar el valor del nuevo bit más significativo, y el de cero (*Z*) pasa a valer *1*, si el nuevo valor del registro es 0000_H.

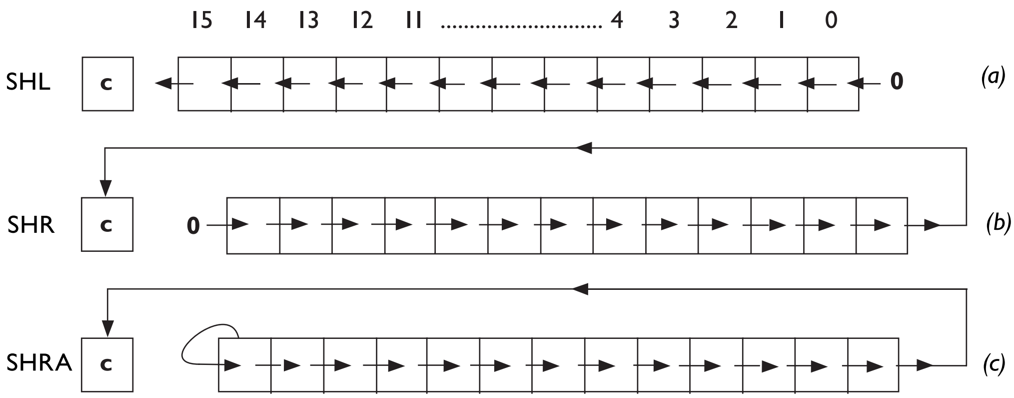


Figura 8.24. *Desplazamiento de los bits de un registro provocado por las instrucciones: (a) SHL, (b) SHR, y (c) SHRA.*

EJEMPLO 8.10

¿Cómo lograría desplazar a la izquierda los bits del registro rD?

Con la instrucción será SHL rD, o, en código máquina, 9D00.

En la Figura 8.25 puede verse un esquema de las operaciones realizadas para la ejecución de esta instrucción. Después de captar la instrucción, el registro IR contendrá la instrucción (9D00), y PC la dirección de la siguiente instrucción a ejecutar, suponiendo que la instrucción estuviese en la dirección 003F, el nuevo valor de PC será 0040. Como el codop es H'9 = B'1001, se tiene la instrucción SHL, cuyo formato es del tipo F1, con lo que el único registro referenciado en la instrucción es: rx = rD. Teniendo en cuenta que el contenido inicial de rD es E03B, el resultado del desplazamiento será:

		C ↓		
rD inicial →	E03B	-	1110 0000 0011 1011	
rD final →		1	1100 0000 0111 011***0***	→ C076

Es decir, el nuevo valor de rD será: C076, y los biestables indicadores: Z = 0, S = 1 y C = 1; V conservará su valor previo.

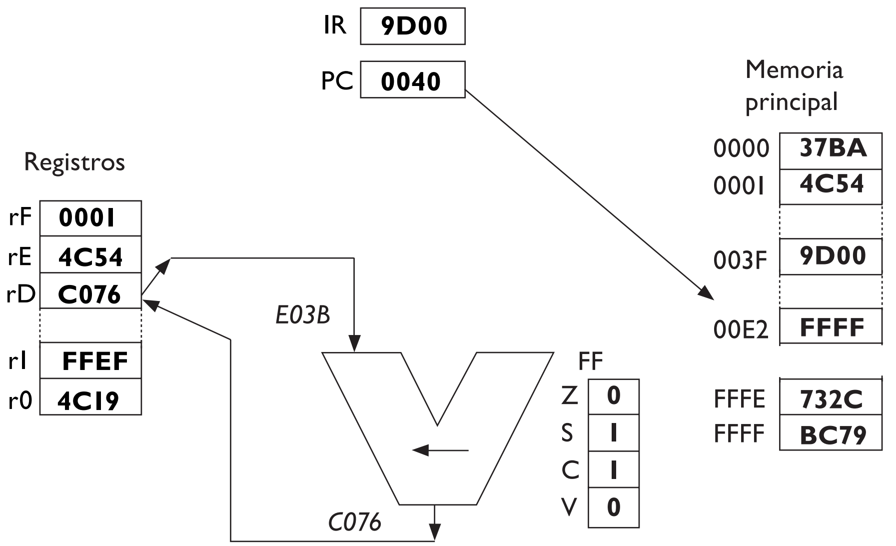


Figura 8.25. *Ejemplo de ejecución de una instrucción desplazamiento a izquierda (SHL rD).*

SHR (desplazamiento a derecha)

Instrucción:	desplazamiento a la derecha de los bits de un registro.
Código de operación:	1010 (A en hexadecimal).
Tipo de formato:	F1 (Figura 8.12).
Instrucción en nemónico:	SHR rx.
Descripción de función a realizar:	C ← rx(0), rx(i) ← rx(i + 1), i = 0,...,14; rx(15) ← 0

Esta instrucción, Figura 8.23b, desplaza los bits del registro *rx*, especificado en la propia instrucción, una posición a la derecha. El bit menos significativo pasa al biestable de acarreo (*C*) y el bit más significativo se hace *0*. El biestable de signo (*S*) pasa a tomar el valor del nuevo bit más significativo, y el de cero (*Z*) pasa a valer *1*, si el nuevo valor del registro es $H'0000_H$.

EJEMPLO 8.11

Supóngase que se desea desplazar a la derecha los bits del registro rD. ¿Cómo lo realizaría?

Utilizando la instrucción SHR rD, o, en código máquina, AD00.

En la Figura 8.26 puede verse un esquema de las operaciones realizadas para la ejecución de esta instrucción. Después de captar la instrucción, el registro IR contendrá la instrucción (AD00), y PC la dirección de la siguiente instrucción a ejecutar, suponiendo que la instrucción estuviese en la dirección 0040, el nuevo valor de PC será 0041. El codop es H'A = B'1010 identifica a la instrucción SHR, cuyo formato es del tipo F1, con lo que el único registro referenciado en la instrucción es: rx = rD. Teniendo en cuenta que el contenido inicial de rD es C076, el resultado del desplazamiento será:

		C ↓		
rD inicial →	C076	–	1100 0000 0111 0110	
rD final →		0	***0***110 0000 0011 1011	→ 603B

Es decir, el nuevo valor de rD será: 603B, y los biestables indicadores tomarán los siguientes valores: Z = 0, S = 0 y C = 0; y V conservará su valor previo.

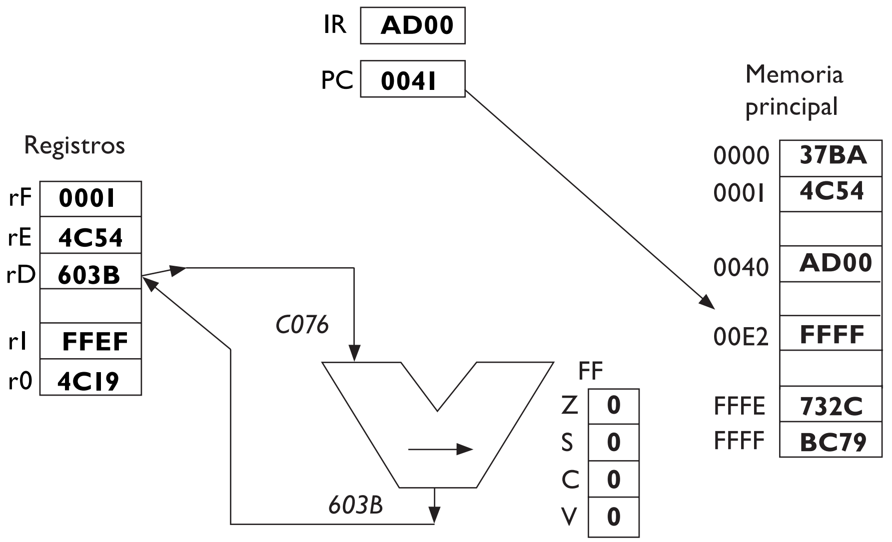


Figura 8.26. *Ejemplo de ejecución de una instrucción desplazamiento a derecha (SHR rD).*

SHRA (desplazamiento aritmético a derecha)

Instrucción: desplazamiento aritmético a la derecha de los bits de un registro.
Código de operación: 1011 (B en hexadecimal).
Tipo de formato: F1 (Figura 8.12).
Instrucción en nemónico: SHRA rx.
Descripción de función a realizar: C ← rx(0), rx(i) ← rx(I + 1), i = 0,…,14.

Esta instrucción, Figura 8.24c, desplaza los bits del registro *rx*, especificado en la propia instrucción, una posición a la derecha. El bit menos significativo pasa al biestable de acarreo (*C*) y el bit más significativo conserva su valor original. El biestable de signo (*S*) pasa a tomar el valor del bit más significativo, y el de cero (*Z*) pasa a valer 1, si el nuevo valor del registro es *H'0000*.

EJEMPLO 8.12

¿Cómo se pueden desplazar aritméticamente a la derecha los bits del registro rD?

Utilizando la instrucción será SHRA rD, o, en código máquina, BD00.

En la Figura 8.27 puede verse un esquema de las operaciones realizadas para la ejecución de esta instrucción. Después de captar la instrucción, el registro IR contendrá la instrucción (BD00), y PC la dirección de la siguiente instrucción a ejecutar, suponiendo que la instrucción estuviese en la dirección 0041, el nuevo valor de PC será 0042. El codop es H'B = B'1011 identifica a la instrucción SHR, cuyo formato es del tipo F1, con lo que el único registro referenciado en la instrucción es: rx = rD. Teniendo en cuenta que el contenido inicial de rD era C076, el resultado del desplazamiento será:

		C ↓		
rD inicial →	C076	–	1100 0000 0111 0110	
rD final →		0	***1***110 0000 0011 1011	→ E03B

Es decir, el nuevo valor de rD será: E03B, y los biestables indicadores tomarán los siguientes valores: Z = 0, S = 1 y C = 0; y V conservará su valor previo.

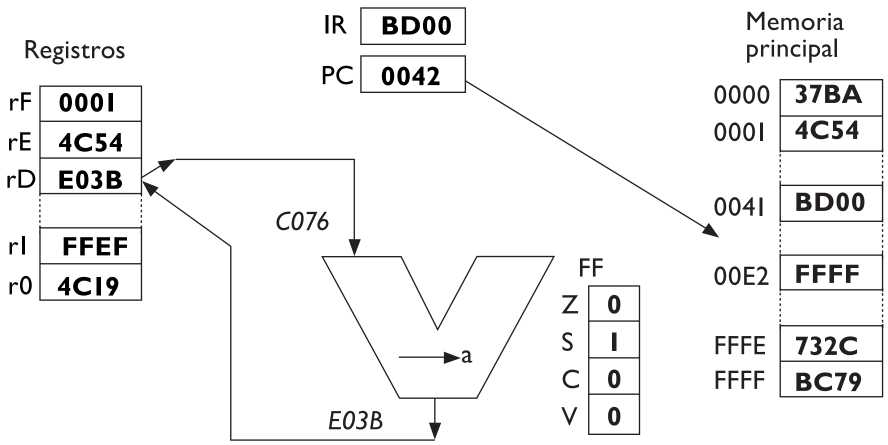


Figura 8.27. *Ejemplo de ejecución de una instrucción desplazamiento aritmético a derecha (SHR rD).*

B- (salto)

Instrucción:	ruptura de secuencia (salto).
Código de operación:	1100 (C en hexadecimal).
Tipo de formato:	F2 (Figura 8.12).
Instrucción en nemónico:	BR (*salto incondicional*).
	BZ (*salto si Z = 1*).
	BS (*salto si S = 1*).
	BC (*salto si C=1*).
	BV (*salto si V = 1*).
Descripción de función a realizar:	Si cnd es incondicional o si se cumple la condición, entonces PC ← rD.

Esta instrucción provoca un salto a la instrucción cuya dirección se encuentra en el registro *rD*. El campo *cnd* que incluye el formato de esta instrucción (formato *F2*, Figura 8.12) especifica la condición de salto, según los códigos de la Tabla 8.4. Es decir, dependiendo de si el valor del biestable indicador especificado por *cnd* es 1 después de la última operación realizada en la *ALU* (con una instrucción *ADDS*, *SUBS*, *NAND*, *SHL, SHR* o *SHRA*, Tabla 8.3), se realiza o no el salto. Si el salto es incondicional (instrucción *BR*, *cnd = 0000*), se efectuará siempre el salto, independientemente del valor de los biestables indicadores.

EJEMPLO 8.13

Indicar las instrucciones para provocar un salto incondicional a la dirección 0041.

Las instrucciones para lograr este objetivo son:

Instrucciones máquina			Explicación
Nemónico	**Hex**	**Binario**	
LLI rD, 41	2D41	0010 1101 0100 0001	Cargar rD con 0041
BR	C0AA	1100 0000 1010 1010	Salto incondicional

Obsérvese que antes de dar la instrucción de salto hay que cargar en rD la dirección de salto, acción que se puede realizar con una instrucción LLI. Como el formato F2 no utiliza el último byte de la instrucción, en éste se puede poner cualquier valor ya que es ignorado por la unidad de control; en la instrucción BR hemos puesto arbitrariamente AA.

Suponemos que la instrucción de salto se encuentra en la posición 004F. En la Figura 8.28 puede verse un esquema de las operaciones realizadas para la ejecución de la instrucción BR. Después de captar la instrucción, el registro IR contendrá la instrucción (C0AA). El codop es H'C=B'1100 identifica a la instrucción B-, cuyo formato es del tipo F2, con lo que la condición de salto se obtiene del segundo cuarteto de la instrucción; en este caso cnd = 0000; es decir se trata de un salto incondicional, con lo que sencillamente la unidad de control cambia el valor que hubiese en el contador de programa (PC) por el contenido de rD, en este caso 0041. De esta forma la próxima instrucción a ejecutar será la ubicada en 0041, como se pretendía con la instrucción.

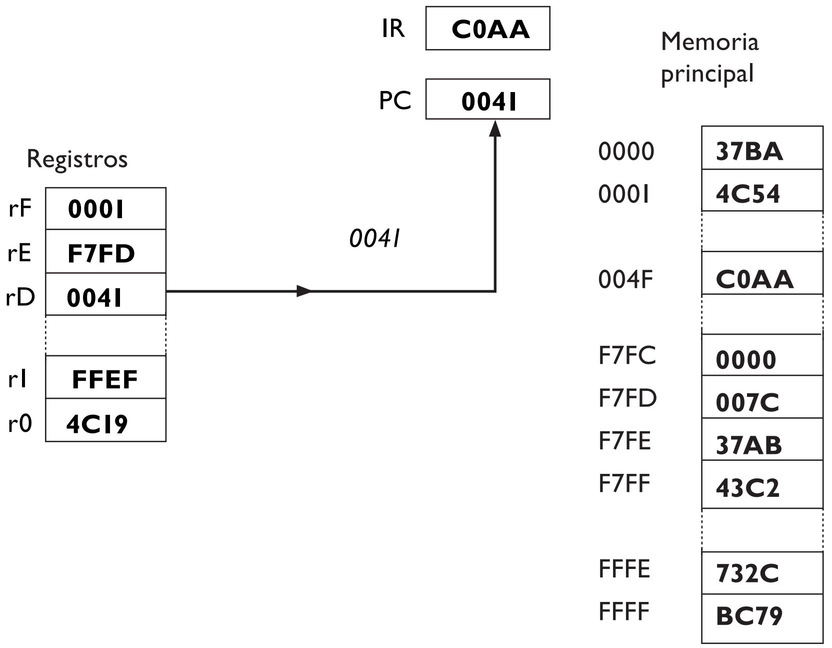


Figura 8.28. *Ejemplo de ejecución de una instrucción de salto incondicional (BR).*

CALL- (llamada a subrutina)

Instrucción:	llamada a subrutina.
Código de operación:	1101 (D en hexadecimal).
Tipo de formato:	F2 (Figura 8.12).
Instrucción en nemónico:	CALLR (llamada incondicional).
	CALLZ (*llamada si Z* = *1*).
	CALLS (*llamada si S* = *1*).
	CALLC (*llamada si C* = *1*).
	CALLV (*llamada si V* = *1*).
Descripción de función a realizar:	Si cnd es incondicional o se cumple la condición, entonces rE ← rE − 1, M(rE) ← PC, PC ← rD.

Esta instrucción provoca una llamada a la subrutina cuya dirección se encuentra en el registro *rD.* El campo *cnd* que incluye el formato de esta instrucción (formato F2, Figura 8.12) especifica la condición de salto, según los códigos de la Tabla 8.4. Es decir, dependiendo de si el valor del biestable indicador especificado por *cnd* es 1 después de la última operación realizada en la *ALU* (con una instrucción *ADDS*, *SUBS*, *NAND*, *SHL, SHR* o *SHRA*, Tabla 8.3), se realiza o no la llamada. Si la llamada es incondicional (instrucción *BR*, *cnd = 0000*), se efectuará siempre la llamada, independientemente del valor de los biestables indicadores. Recuérdese (Sección 7.3.2) que en la ejecución de las instrucciones de llamada a subrutinas, previamente a cambiar el valor del contador de programa por la dirección de salto (*PC ← rD*, en el caso de CODE-2), debe guardarse el valor previo del contador de programa en la pila (en el caso de CODE-2, *rE ← rE – 1, M(rE) ← PC,* ya que *rE* actúa como puntero de pila).

EJEMPLO 8.14

Supóngase que se desea realizar llamar a la subrutina que comienza en la dirección C03B en el caso de que el resultado de la última operación realizada en la ALU hubiese sido cero; es decir, si Z=1. ¿Qué instrucciones habría que utilizar?

Las instrucciones para lograr este objetivo son:

Instrucción			**Explicación**
Nemónico	**Hex**	**Binario**	
LI rD,3B	2D3B	0010 1101 0011 1011	Cargar rD con 003B
LHI rD,C0	3DC0	0011 1101 1100 0000	Cambiar primer byte de rD por C0
CALLZ	D100	1101 0001 0000 0000	Llamada a subrutina si Z = 1

Obsérvese que antes de dar la instrucción de salto hay que cargar en rD la dirección de salto (C03B), acción que se realiza con las instrucciones de carga inmediata LLI y LHI. El segundo cuarteto de la instrucción CALL es 0001, ya que la condición de llamada es que Z = 1 (véase Tabla 8.4).

Suponemos que la instrucción de salto se encuentra en la posición 0042. En la Figura 8.29 puede verse un esquema de las operaciones realizadas para la ejecución de la instrucción BR. Después de captar la instrucción, el registro IR contendrá la instrucción (D100). El codop es H'D = B'1101 identifica a una instrucción CALL-, cuyo formato es del tipo F2, con lo que la condición de salto se obtiene del segundo cuarteto de la instrucción; en este caso cnd = 0001; es decir se trata de una llamada condicionada al valor de Z. Supongamos que Z = 1, en este caso la unidad de control decrementa en una unidad el valor del puntero de pila (rE) y almacena en M(rE) el valor del contador de programa (es decir, 0043), y después sencillamente cambia el valor previo del contador de programa (0043) por el contenido de rD, en este caso C03B. De esta forma la próxima instrucción a ejecutar será la ubicada en C03B, habiéndose producido por tanto la llamada a la subrutina.

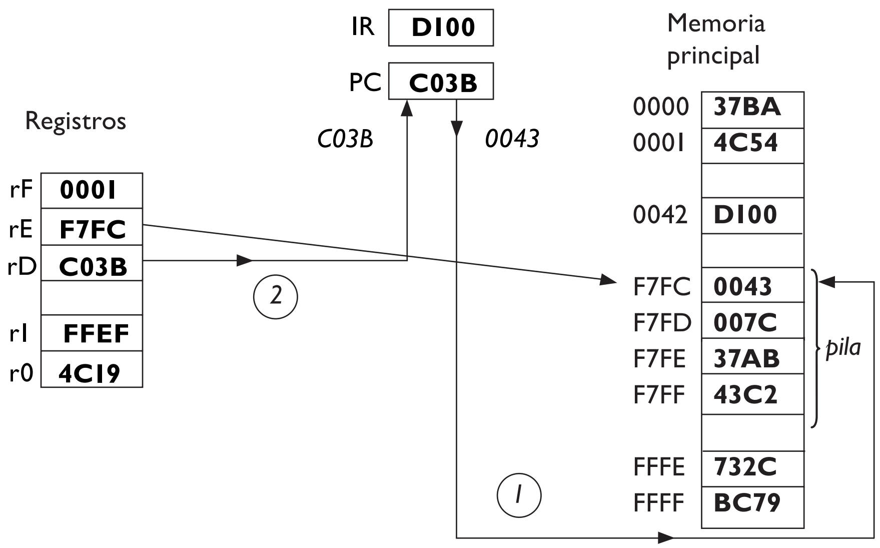


Figura 8.29. *Ejemplo de ejecución de una instrucción de llamada a subrutina si Z = 1 (CALLZ).*

RET (retorno de subrutina)

Instrucción: retorno de subrutina.
Código de operación: 1110 (E en hexadecimal).
Tipo de formato: F0 (Figura 8.12).
Instrucción en nemónico: RET.
Descripción de función a realizar: PC ← M(rE); rE ← rE + 1.

Esta instrucción se incluye al final de una subrutina, y provoca un retorno al último programa que la llamó; concretamente a la instrucción siguiente al *CALL*-. Recuérdese (Sección 7.3.2) que en la ejecución de una instrucción de retorno, sencillamente debe cambiarse el contenido del contador de programa por el valor de la cabecera de la pila. En el caso de CODE-2 la dirección de la cabecera de la pila está en rE, con lo que la operación que realizará la unidad de control es: *PC ← M(rE)*. Además hay que actualizar el valor del puntero de pila; es decir, el procesador hace *rE ← rE + 1*.

EJEMPLO 8.15

Supóngase que, después de ejecutada la subrutina que comienza en C03B del Ejemplo 8.14, se desea retornar al programa que la llamó. ¿Cómo lograrlo?

Con la instrucción RET, o, en código máquina, EFFF.

Suponemos que la instrucción de retorno se encuentra en la posición C0F7. En la Figura 8.30 puede verse un esquema de las operaciones realizadas para la ejecución de la instrucción RET. Después de captar la instrucción, el registro IR contendrá la instrucción (EFFF). El codop es H'F = B'1110, identifica a una instrucción RET, cuyo formato es del tipo F0, con lo que no se considera ningún campo adicional (los valores FFF no son analizados por la unidad de control). La unidad de control hace que se lea el valor de la dirección de retorno, que se encuentra en la cabecera de la pila; es decir, en la posición rE (F7FC) de memoria y este valor (0043) se carga en el contador de programa. Después se actualiza el valor del puntero de pila (rE), incrementándolo en una unidad: pasa de F7FC a F7FD. En definitiva, como resultado se esta instrucción, la próxima instrucción a ejecutar será la ubicada en 0043, habiéndose producido en consecuencia el retorno a la dirección siguiente desde donde se llamó a la subrutina.

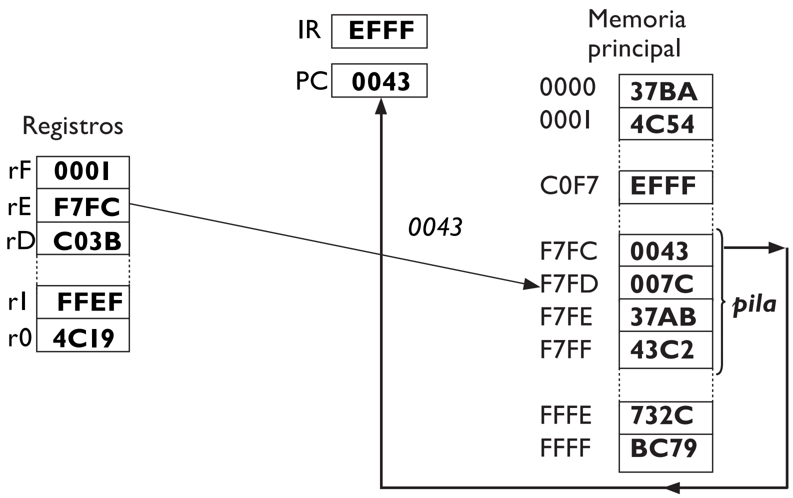


Figura 8.30. *Ejemplo de ejecución de una instrucción de retorno de subrutina (RET).*

HALT (parada)

Instrucción: *parada.*
Código de operación: 1111 (F en hexadecimal).
Tipo de formato: F0 (Figura 8.12).
Instrucción en nemónico: HALT.

Esta instrucción provoca que CODE-2 entre en un **estado de espera**, deteniéndose su funcionamiento justo antes de la fase de captación de la siguiente instrucción, tal y como se indica en la Figura 8.31. Puede continuarse la ejecución del programa detenido pulsando la tecla CONTINUAR del panel de control (Sección 8.6).

El uso de esta instrucción es de interés, por ejemplo, como final de un programa, o cuando se desea detener a CODE-2 después de haber dado un resultado por un puerto de salida para que el programador u operador de CODE-2 confirme la visualización de dicho resultado (véase Sección 8.3.4).

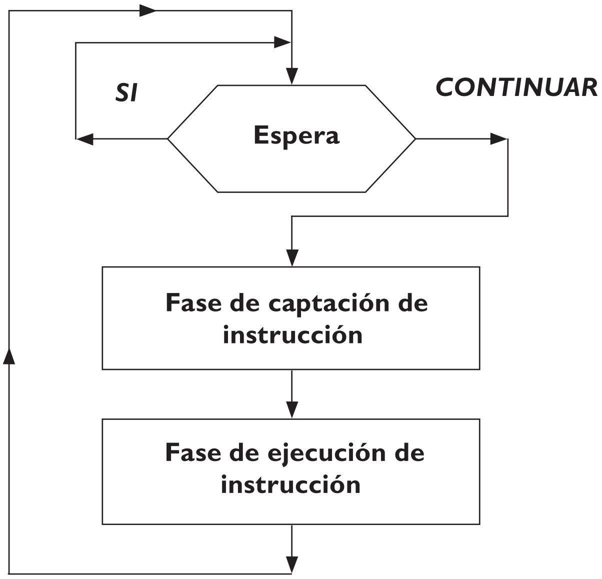


Figura 8.31. *El estado de espera se produce antes del inicio de la captación de la siguiente instrucción.*

8.3.4 Algunos trucos de programación

La realización de un programa que implemente un algoritmo consiste en la descripción de éste utilizando únicamente las instrucciones que ofrece el repertorio de instrucciones del lenguaje de programación a utilizar. Con otras palabras, y refiriéndonos al caso de CODE-2, para realizar cualquier programa *únicamente* podemos utilizar las 16 instrucciones de la Tabla 8.5. Utilizando estas instrucciones con ingenio debemos poder realizar cualquier programa, por complicado que sea.

Para dar una idea de las posibilidades del repertorio de instrucciones de CODE-2, en esta sección incluimos algunas técnicas frecuentemente usadas. Como complemento en la Sección 8.3.5 describiremos algunos ejemplos de programas completos.

Puesta a cero o a uno de un registro

Con mucha frecuencia los programas utilizan los valores 0 y 1 y es aconsejable introducir estos valores en dos registros, por ejemplo el *R0* y *R1*, respectivamente. Esto puede hacerse con las siguientes instrucciones:

Instrucción máquina			Explicación
Nemónico	**Hexadecimal**	**Binario**	
LLI r0,00	2000	0010 0000 0000 0000	Cargar r0 con H'0000
LLI r1,01	2101	0010 0001 0000 0001	Cargar r1 con H'0001

Copiar el contenido de un registro en otro

Se puede llevar el contenido de un registro a otro utilizando una instrucción de suma. Suponiendo que en *r0* hay el valor *0000*, el valor de *r4* se puede pasar a *rE* así:

Instrucción máquina			Explicación
Nemónico	**Hexadecimal**	**Binario**	
ADDS rE,r4,r0	6E40	0110 1110 0100 0000	rE ← r4 + 0

Los biestables indicadores quedan modificados de acuerdo con el valor de *r0*.

Detectar si un número es cero o negativo

Se puede detectar si un número, en un registro determinado es cero, activando los biestables indicadores con su valor. Esto puede realizarse, sin más que sumarle (o restarle) el valor 0. Así, supóngase que se desea saltar a la instrucción cuya dirección está en *rD* si el valor del registro *r4* es cero, y que en *r0* está almacenado el valor *0000*; el código máquina para lograr ese objetivo sería:

Instrucción máquina			Explicación
Nemónico	**Hexadecimal**	**Binario**	
ADDS r4,r4,r0	6440	0110 0100 0100 0000	Sumar 0 al número para activar los biestables
BZ	C100	1100 0001 0000 0000	Saltar si el biestable de cero (Z) se activa

De igual forma se puede detectar si *r4* es negativo; sin más que sustituir la última instrucción (*BZ*) por la *BS*.

Comparar dos números

La comparación de un número consiste en activar los biestables indicadores de acuerdo con la diferencia de sus valores. Así, por ejemplo, si se desease saltar a la instrucción ubicada en *F300* si *r5 = r4*, a la ubicada en *547C* si *r5 < r4* y a la ubicada en *737C* si *r5 > r4*, se podrían utilizar las siguientes instrucciones:

Instrucción máquina			Explicación
Nemónico	**Hexadecimal**	**Binario**	
SUBS rF,r5,r4	7F54	0111 1111 0101 0100	Comparar r5 con r4

	Instrucción máquina		Explicación
Nemónico	**Hexadecimal**	**Binario**	
LLI rD,00	2D00	0010 1101 0000 0000	Dirección de salto si son iguales
LHI rD,F3	3DF3	0011 1101 1111 0011	
BZ	C100	1100 0001 0000 0000	Saltar si son iguales (Z = 1)
LLI rD,7C	2D7C	0010 1101 0111 1100	Dirección de salto si r4 > r5
LLH rD,54	3D54	0011 1101 0101 0100	
BS	C200	1100 0010 0000 0000	Saltar si r4 > r5 (S = 1)
LLH rD,73	3D73	0011 1101 0111 0011	Dirección de salto si r4 < r5
BR	C000	1100 0000 0000 0000	Salto incondicional

Obsérvese que:

- Con la instrucción de resta activamos los biestables indicadores, de acuerdo con la diferencia de los dos registros a comparar.
- Las instrucciones *LLI* y *LHI* no modifican los valores de los biestables indicadores (Tabla 8.3).

No hacer nada (instrucción no operativa)

A veces interesa incluir en un programa instrucciones que no realicen ninguna operación; esto se puede hacer sumando a un registro cualquiera el valor cero:

	Instrucción máquina		Explicación
Nemónico	**Hexadecimal**	**Binario**	
ADDS r4,r4,r0	6440	0110 0100 0100 0000	r4 ← r4 + 0

Ahora bien, esta instrucción no conserva el estado de los biestables indicadores.

Las instrucciones no operativas se utilizan frecuentemente para hacer rutinas de retardo; éstas tienen por objeto consumir un determinado intervalo de tiempo. La instrucción *ADDS* consume 7 ciclos de reloj (Tabla 6.5); es decir, si la frecuencia de reloj de CODE-2 fuese de 10 MHz, su ejecución tardaría $7/(10 \cdot 10^6) = 0{,}7\mu s$. Con un bucle de instrucciones puede realizarse una rutina de un retardo concreto (ver Ejercicio 6.13).

Contadores ascendentes y descendentes

Suponiendo que en *r1* hemos memorizado el *0001*, es fácil incrementar o decrementar en *1* el valor de cualquier registro sin más que utilizar instrucciones de suma y de resta:

	Instrucción máquina		Explicación
Nemónico	**Hexadecimal**	**Binario**	
ADDS r4,r4,r1	6441	0110 0100 0100 0001	Incremento de r4 en 1
SUBS r5,r5,r1	7551	0111 0101 0101 0001	Decremento de r5 en 1

Implementación de un puntero que recorra una tabla
Supóngase que se desea operar con los datos de una tabla que está en la memoria principal, y cuya dirección de comienzo se encuentra en rA y la dirección final en rB. Utilizando el registro r2 como puntero de la tabla (PT), el recorrido es fácil realizarlo con las siguientes instrucciones:

Dcción. Memoria	Instrucción máquina		Explicación
	Nemónico	Hexadecimal	
00A0	ADDS r2,rA,r0	62A0	Establecer el valor inicial del puntero
00A1	LLI rD,B6	2DB6	Llevar a rD dirección del final de programa
00A2	SUBS rF,rB,r2	7FB2	Ver si el puntero sobrepasa el final de tabla
00A3	BS	C200	Si se sobrepasa ir al final saltar a 00B6
00A4	ADDS rD,r2,r0	6D20	Llevar el puntero al registro de dirección
00A5	LD r3,[00]	0300	Llevar dato de la tabla a r3
(00A6 a 00B2: operaciones con el dato de la tabla, que se encuentra en r3)			
00B3	ADD r2,r2,r1	6221	Incrementar el puntero de la tabla
00B4	LLI rD,A1	2DA1	Dirección de salto
00B5	BR	C000	Salto incondicional a 00A1
00B6	HLTA	F000	Fin

Como se observa en primer lugar se comprueba si se ha llegado al final de la tabla (instrucciones en *00A1*, *00A2* y *00A3*), después se lleva el dato de la tabla a r3 (instrucciones en *00A4* a *0A59*), después se hacen las operaciones que corresponda con el dato de la tabla (instrucciones de la *00A6* a la 00B2), se incrementa el puntero de la tabla (instrucción en *00B3*), y por último (instrucciones en *00B4* y *00B5*) se salta al inicio del bucle, comprobándose allí si se ha llegado al final de la tabla.

Entradas y salidas
Según se indicó en la Sección 6.3.3, en la versión implementada de CODE-2 se utiliza como entrada un teclado hexadecimal (*IP1*) y como salidas dos conjuntos de visualizadores (*OP1* y *OP2*).

Al ejecutar una instrucción de salida, inmediatamente aparece en el visualizador del puerto de salida la información del registro correspondiente, y ésta permanece allí inalterada hasta que se vuelva a dar otra salida por el mismo puerto. Si se dan salidas sucesivas por un puerto (por ejemplo se están visualizando los datos de una tabla) los datos del visualizador cambiarán tan rápidamente que el operador no podrá verlos. Puede evitarse este problema incluyendo, después de la instrucción de salida (*OUT*) una instrucción de espera (*HALT*). De esta forma después de cargarse el puerto de salida con el valor del registro se detiene CODE-2, manteniéndose todos sus contenidos internos fijos, y en particular los de los puertos de salida. El operador, una vez percatado del valor de salida, puede hacer que continúe el programa sin más que pulsar la tecla *CONTINUAR* del panel de control (Sección 6.3.6).

Problemas parecidos se plantean en las instrucciones de entrada. En efecto, las instrucciones de entrada captan inmediatamente el valor que hubiese en *IP1*, de forma que es muy posible que el operador no hubiese podido adelantarse al programa introducido previamente a la ejecución de la instrucción de entrada el dato requerido. Este problema se plantea claramente

si se solicitan, por ejemplo, dos datos sucesivos: el operador tendría que introducirlos en un intervalo de tiempo menor que el transcurrido entre la ejecución de las instrucciones de las dos entradas sucesivas, lo cual resultará imposible. En otras palabras, es necesario que exista una sincronización entre el momento que el dato es tecleado correctamente por el operador, y el instante en que el dato es captado por la instrucción. Este problema, como en el caso de la entradas, también se puede solucionar con instrucciones *HALT*; pero una solución más cómoda para el operador es hacer preceder a las instrucciones de entrada instrucciones de salida de forma que el operador pueda identificar el dato requerido, tal y como se hace en el siguiente ejemplo.

EJEMPLO 8.16

Indicar unas posibles instrucciones para introducir a través del teclado unos datos para inicialización de los registros r0, r1 y r2.

Instrucción máquina		Explicación
Nemónico	Hexadecimal	
LLI r3,AA	23AA	Byte bajo a visualizar
LHI r3,0D	330D	Byte alto a visualizar
OUT OP01,r3	5301	Salida del primer mensaje “0DAA”
IN r0,IP01	4001	Entrada del primer valor, a almacenar en r0
LHI r3,1D	331D	Byte alto a visualizar
OUT OP01,r3	5301	Salida del segundo mensaje “1DAA”
IN r1,IP01	4101	Entrada del segundo valor, a almacenar en r1
LHI r3,2D	332D	Byte alto a visualizar
OUT OP01,r3	5301	Salida del tercer mensaje “2DAA”
IN r2,IP01	4201	Entrada del tercer valor, a almacenar en r2

Durante la ejecución del programa y cuando CODE-2 se detenga, el operador tiene que realizar lo siguiente:

1. *Cuando en el visualizador OP1 aparezca 0DAA, teclear el valor a almacenar en r0, y pulsar CONTINUAR.*
2. *Cuando en el visualizador OP1 aparezca 1DAA, teclear el valor a almacenar en r1, y pulsar CONTINUAR.*
3. *Cuando en el visualizador OP1 aparezca 2DAA, teclear el valor a almacenar en r2, y pulsar CONTINUAR.*

De esta forma, el usuario cuando vea el mensaje debe teclear el dato a introducir en el registro, y después pulsar CONTINUAR.

Detectar si un número es par

Como ejemplo de otras posibilidades del lenguaje máquina de CODE-2 a continuación se indican tres formas posibles de comprobar si un número es par. Por concretar, se trata de saltar a la instrucción en la posición A73B si el número que esté almacenado en el registro r3 es par, y a la B73C si es impar.

La primera forma es implementando el algoritmo del Ejemplo 2.13 del Capítulo 2, que, por comodidad, volvemos a incluir en el organigrama de la Figura 8.32.

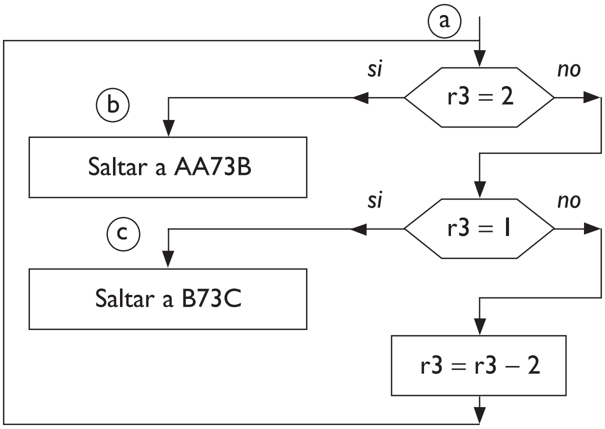


Figura 8.32. *Un algoritmo para comprobar si un número es par (Ejemplo 2.13.)*

Suponiendo que en r0 está almacenado *0000* y que en *r1* está almacenado *0001*, las instrucciones que implementan el algoritmo anterior son las siguientes:

Dirección de memoria	Instrucción máquina		Explicación
	Nemónico	Hexadecimal	
00A0	LLI rD,3B	2D3B	Dirección baja de primer salto
00A1	LHI rD,A7	3DA7	Dirección alta de primer salto
00A2	SUBS rF,r3,r0	7F30	Para comprobar si r3 = 0
00A3	BZ	C100	Saltar a (b) si r3 = 0
00A4	LLI rD,3C	2D3C	Dirección baja de segundo salto
00A5	LHI rD,B7	3DB7	Dirección alta de segundo salto
00A6	SUBS r3,r3,r1	7331	Para comprobar si r3 = 1
00A7	BZ	C100	Saltar a (c) si r3 = 1
00A8	SUBS r3,r3,r1	7331	Restar otra vez 1 a r3
00A9	LLI rD,A0	2DA0	Dirección baja del tercer salto
00AA	LHI rD,00	3D00	Dirección alta del tercer salto
00AB	BR	C000	Salto incondicional a (a)

El algoritmo anterior presenta los siguientes inconvenientes:

1. El tiempo de ejecución es elevado (12 instrucciones) y depende del valor de r3. Concretamente el número de iteraciones aproximadamente es (r3 + 1)/2; es decir, cuanto mayor es r3 mayor es el tiempo de detección de la paridad del número.
2. Sólo vale para números positivos.

El segundo y tercer método se basan en comprobar si el último bit del número binario es 0, en cuyo caso el número sería par, o es 1, en cuyo caso el número sería impar.

Para comprobar si el último número es 0, con el segundo método utilizamos la operación lógica *NAND*. Es fácil comprobar, con ayuda de la Tabla 8.7, que si hacemos la operación lógica *NAND* de cualquier número con *H'0001* el resultado será *FFFF*, si el número acaba en 0 (es decir, si es par), y *FFFE*, si el número acaba en 1 (es decir, si es impar). Con las siguientes instrucciones se implementa el procedimiento anterior, donde en r4 se almacena la máscara *FFFF* y se ha supuesto que en *r1* está almacenado *0001*.

Dirección Memoria	Instrucción máquina		Explicación
	Nemónico	Hexadecimal	
00A0	LLI r4,FF	24FF	Parte baja de la máscara
00A1	LHI r4,FF	34FF	Parte alta de la máscara
00A2	NAND r5,r3,r1	8531	Operación NAND entre dato y H'0001
00A4	LLI rD,3B	2D3B	Dirección baja de salto si par
00A5	LHI rD,A7	3DA7	Dirección alta de salto si par
00A6	SUBS rF,r5,r4	7F54	Comparar con la máscara
00A7	BZ	C100	Saltar a A73B si es par
00A4	LLI rD,3C	2D3C	Dirección baja de salto si impar
00A5	LHI rD,B7	3DB7	Dirección alta de salto si impar
00AB	BR	C000	Salto incondicional a B73C por ser impar

El tercer método sugerido comprueba si el último bit de r3 es cero, haciendo un desplazamiento del número a la derecha (SHR) y efectuando un salto a B73C si C = 1 (Figura 8.24), ya que en este caso el número sería impar. Las instrucciones serían las siguientes:

Dirección Memoria	Instrucción máquina		Explicación
	Nemónico	Hexadecimal	
00A0	LLI rD,3C	2D3C	Dirección baja de salto si impar
00A1	LHI rD,B7	3DB7	Dirección alta de salto si impar
00A2	SHR r3	A300	Desplazar r3 a la derecha
00A4	BC	C300	Saltar a B73C si r3 era impar
00A5	LLI rD,3B	2D3B	Dirección baja de salto si par
00A6	LHI rD,A7	3DA7	Dirección alta de salto si par
00A7	BR	C000	Salto incondicional a A733 por ser par

Puede observarse que con el tercer procedimiento es el más eficiente.

8.3.5 Ejemplos de programas

Según se indicó en la sección anterior, el repertorio de instrucciones de CODE-2 es muy limitado no obstante pueden programarse con facilidad numerosos algoritmos. A título de ejemplo en está sección incluimos tres ejemplos. En los ejercicios de final del capítulo se incluyen otros, que pueden complementarse con los planteados en la página web del libro [webCODE].

En general, para realizar un programa de CODE-2, es conveniente seguir los siguientes pasos:

a) Describir el algoritmo a implementar por medio de un organigrama o pseudocódigo. Las reglas para representar algoritmos se presentaron en la Sección 3.2.
b) Efectuar una asignación de registros y de posiciones de memoria para los parámetros y variables utilizadas en el programa. Asignar la dirección de inicio del programa.
c) Redactar el programa en nemónicos.
d) Codificar las instrucciones en código máquina (hexadecimal o binario)
e) Documentar adecuadamente el programa, indicando los pormenores para uso del programa: dirección a partir de la que se debe cargar el programa, datos que solicita el programa durante su ejecución, significado de los resultados de salida, etc.

Programa de carga en memoria

Realizar un programa para CODE-2 que cargue a partir de una posición inicial de memoria, *DC*, dada a través del teclado, la información que se vaya dando sucesivamente también por el teclado.

Descripción del algoritmo

El algoritmo primero lee del teclado (IP1) el valor de *DC* (dirección de comienzo de la carga), y posteriormente solicita los valores a almacenar. Para facilitar el uso del programa, previamente a la instrucción de entrada de *DC*, en el visualizador *OP1* se muestra el mensaje *DCDC*, y previamente a las instrucciones de entrada de la información a memorizar, en *OP1* se muestra la dirección donde se almacenará la información que a continuación teclee el usuario.

En la Figura 8.33 puede verse un organigrama del algoritmo a implementar. Al comienzo del programa se incluye un conjunto de instrucciones que sirven para especificar los parámetros iniciales del programa. Las variables que se utilizan son:

DC: dirección de carga, y
PM: palabra a memorizar

Asignación de registros y de memoria

En la Tabla 8.8 se muestra la asignación de registros y de posiciones de memoria realizada para los parámetros y variables que intervienen en el programa.

Tabla 8.8. *Asignación de registros y de memoria del programa de carga en memoria.*

Parámetro o variable	Registro	Posición de memoria	Comentario
0001	r1		Para el contador
0000	r0		Para pasar el valor de un registro a otro
DCDC	rA		Mensaje para solicitar la dirección inicial de carga
DC	r2		Dirección de carga
PM	r3		Palabra a memorizar
Programa		00A0	Dirección de carga del programa

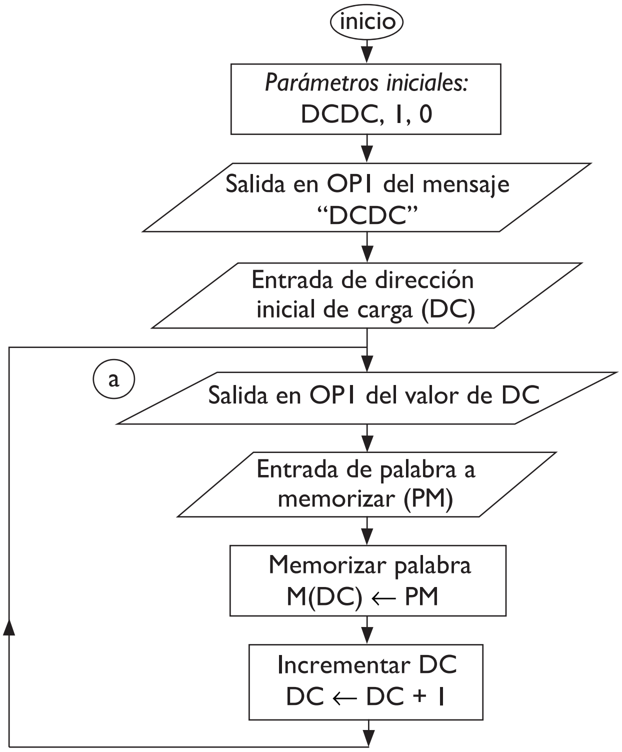


Figura 8.33. *Organigrama del programa de carga en memoria.*

Redacción del programa en nemónicos

Teniendo en cuenta el repertorio de instrucciones de CODE-2 (Tabla 8.5) y la asignación de memoria, el programa en nemónicos es el que se incluye en la tercera columna de la Tabla 8.9. La primera columna especifica simbólicamente las referencias de salto, de acuerdo con la notación utilizada en el organigrama y la segunda columna las direcciones de memoria en que se ubicaría cada instrucción. También en la quinta columna se incluye un breve comentario sobre la instrucción.

Tabla 8.9. *Programa de carga en memoria.*

Rfcia. salto	Dircc. Memoria	Instrucción máquina		Explicación
		Nemónico	Hex	
	00A0	LLI rA,DC	2ADC	Parte baja del mensaje
	00A1	LHI rA,DC	3ADC	Parte alta del mensaje
	00A2	LLI r1,01	2101	Parámetro 0001 en r1
	00A3	LLI r0,00	2000	Parámetro 0000 en r0
	00A4	OUT OP1,rA	5A01	Salida de mensaje DCDC
	00A5	IN r2,IP1	4012	Entrada de dirección de comienzo (DC)

Tabla 8.9. *Programa de carga en memoria (Cont.).*

Rfcia. salto	Dircc. Memoria	Instrucción máquina Nemónico	Hex	Explicación
(a)	00A6	OUT OP1,r2	5201	Salida de dirección de carga
	00A7	IN r3,IP1	4301	Entrada del valor a cargar
	00A8	ADDS rD,r2,r0	6D20	Pasar la direcc. de carga al rD
	00A9	ST 00,r3	1300	Memorizar palabra
	00AA	ADDS r2,r2,r1	6221	Incrementar en 1 dirección de carga
	00AB	LLI rD,A6	2DA6	Cargar en rD dirección de salto
	00AC	BR	C000	Salto incondicional a (a)

Codificación del programa en código máquina

La cuarta columna de la Tabla 8.9 contiene el programa codificado en código máquina (hexadecimal). Conviene indicar que el programa en lenguaje máquina está constituido tan sólo por la segunda columna (dirección de memoria) y la cuarta columna (código máquina) de la tabla indicada. Las otras columnas sirven como ayuda para facilitar la redacción del programa y mejorar su legibilidad

Instrucciones al operador

En la Tabla 8.10 se incluyen los comentarios para el usuario del programa.

Tabla 8.10. *Instrucciones para el uso del programa de carga en memoria.*

Programa de carga en memoria		
Posición inicial del programa	*00A0*	
Datos solicitados por el programa	*IP1*	Cuando aparezca en OP1 el mensaje DCCD, dar la dirección de inicio de la carga (DC) Cuando aparezca en OP1 una dirección de memoria, dar la palabras a carga en ella (PC)
Resultados dados por el programa	*OP1*	Mensaje DCDC (para dar DC) Dirección de carga (para dar PC)

Programa de suma de dos tablas

Se trata de realizar un programa para CODE-2 que sume uno a uno los datos de dos tablas, *T1* y *T2*, y los resultados los incluya en una tercera tabla, *T3*. Suponer que T1 comienza en *d1 = 0040* y *T2* en *d2 = 0080*, y *T3* debe generarse a partir de *d3 = 00C0*. La longitud de las tablas es de *nT = H'20* elementos.

Descripción del algoritmo

El algoritmo, sencillamente, lleva iterativamente el contenido de un elemento T1(i) de la tabla T1 a un registro y el correspondiente elemento *T2(i)* de la tabla *T2* a otro registro. Se suman los dos registros mencionados en un tercer registro, y se memoriza este valor en *T3(i).*

En la Figura 8.34 puede verse un organigrama del algoritmo a implementar. Al comienzo del programa se incluye un conjunto de instrucciones que sirven para especificar los parámetros iniciales del programa. Las variables que se utilizan son:

- número de elementos de la tabla (*nT*),
- valor *T1(i),* valor *T2(i),* valor *T3(i) = T1(i) + T2(i),* e
- índice *i*.

Como las direcciones base (de inicio, *d1, d2* y *d3*) de las tablas son menores de *FF* podemos utilizar direccionamiento indexado para acceder a sus elementos: la dirección de base se incluye en el campo v de la instrucción y utilizamos *rC* como registro índice.

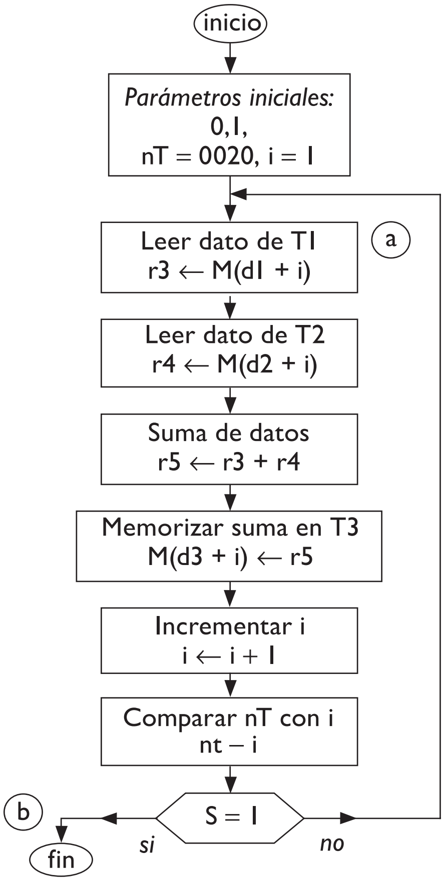


Figura 8.34. *Organigrama del programa de suma de dos tablas.*

Asignación de registros y de memoria

En la Tabla 8.11 se incluye la asignación de registros y de posiciones de memoria para los parámetros, variables y el código máquina del programa.

Tabla 8.11. *Asignación de registros y de memoria del programa de suma de dos tablas.*

Parámetro o variable	Registro	Posición de memoria	Comentario
0000	r0		Para pasar el valor de un registro a otro
0001	r1		Para incrementar el índice i
nT	r2		Número de elementos de la tabla
T1(i)	r3		Elemento de la tabla T1
T2(i)	r4		Elemento de la tabla T2
T3(i)	r5		Elemento de la tabla T3
i	rC		Registro de indexación
Programa		0010	Dirección de carga del programa

Redacción del programa en nemónicos

Teniendo en cuenta el repertorio de instrucciones de CODE-2 (Tabla 8.7) y la asignación de memoria, el programa en nemónicos es el que se incluye en la tercera columna de la Tabla 8.12. La primera columna especifica simbólicamente las referencias de salto, de acuerdo con la notación utilizada en el organigrama y la segunda columna las direcciones de memoria en que se ubicaría cada instrucción. También en la quinta columna se incluye un breve comentario sobre la instrucción.

Tabla 8.12. *Programa de suma de dos tablas.*

Rfcía. salto	Dircc. Memoria	Instrucción máquina		Explicación
		Nemónico	Hex	
	0010	LLI r0,00	2000	Inicializar r0 a 0
	0011	LLI r1,01	2101	Inicializar r1 a 1
	0012	LLI r2,20	2220	Tamaño de tabla: nT=H'20
	0013	LLI rC,01	2C01	Inicialización del índice: i=1
(a)	0014	ADDS rD,rC,r0	6DC0	Pasar el índice al registro de dirección
	0015	LD r3,[40]	0340	Llevar a r3 el elemento de T1
	0016	LD r4,[80]	0480	Llevar a r4 el elemento de T2
	0017	ADDS r5,r3,r4	6534	Suma de elementos de T1 y T2 en r5
	0018	ST [C0],r5	15C0	Almacenar nuevo elemento de T3
	0019	ADDS rC,rC,r1	6CC1	Incrementar el índice
	001A	LLI rD,1F	2D1F	Cargar en rD dirección de salto (final)
	001B	SUBS rF,r2,rC	7F2C	Comparar nT con i
	001C	BS	C200	Si S=1 el programa concluye, salta a (b)
	001D	LLI rD,14	2D14	Cargar en rD dirección de salto (continuar)
	001E	BR	C000	Salto a (a)
(b)	001F	HALT	F000	Final del programa

Codificación del programa en código máquina

La cuarta columna de la Tabla 8.12 contiene el programa codificado en código máquina (hexadecimal). Conviene indicar que el programa en lenguaje máquina está constituido tan sólo por la segunda columna (dirección de memoria) y la cuarta columna (código máquina) de la tabla

indicada. Las otras columnas sirven como ayuda para facilitar la redacción del programa y mejorar su legibilidad

Instrucciones al operador

En la Tabla 8.13 se incluyen los comentarios para el usuario del programa.

Tabla 8.13. *Instrucciones para el uso del programa de suma de dos tablas.*

Programa de carga en memoria	
Posición inicial del programa	**0010**
El programa suma las tablas (de H'20 elementos) ubicadas a partir de las posiciones H'40 y H'80, elemento a elemento, almacenado la tabla resultante a partir de la posición H'C0.	

Programa de ordenación de una tabla de números enteros

Se trata de hacer una rutina para colocar los elementos de una tabla en orden creciente, de menor a mayor. En la posición *H'0010* de memoria se encuentra la dirección de comienzo de la tabla, y en H'0011 su tamaño, y la rutina debe empezar en *H'00A0*.

Descripción del algoritmo

Vamos a utilizar el **algoritmo de ordenación por burbujas**, que consiste en detectar el elemento mayor de la tabla y ubicarlo al final de ella, con lo que éste queda ordenado; después se vuelven a analizar el resto de elementos no ordenados de la tabla y el nuevo máximo se coloca al final, justo inmediatamente antes de la zona previamente ordenada, y así sucesivamente (Figura 8.35). Cada vez que se encuentra un máximo de la zona desordenada, aumenta en una posición la zona ordenada y disminuye en dicha unidad la zona desordenada. Si la tabla contiene nT elementos habrá que recorrer la tabla $nT - 1$ veces. Para seleccionar cada máximo, sencillamente se compara, desde el principio, cada elemento, $T(i)$, $i = 2, ..., nT$, con el anterior, $T(i - 1)$, y si $T(i - 1)$ es mayor que $T(i)$ se intercambian en la memoria, de esta forma al recorrer los elementos no ordenados de la tabla, el último será el nuevo máximo y quedará por tanto en su sitio, como primero de la nueva zona ordenada.

Las variables que vamos a utilizar son (Figura 8.35):

dT: posición inicial de la tabla

pT: puntero de la tabla (posición en memoria del elemento de la tabla al que se accede: dT + i)

pfT: puntero de la posición del último elemento de la zona desordenada, que inicialmente será el último elemento de la tabla; es decir: pF = dT + nT.

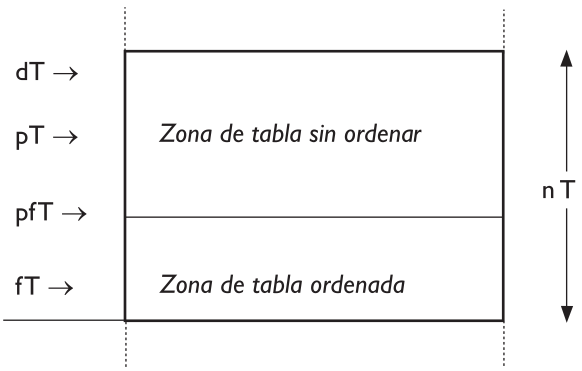


Figura 8.35. *Significado de las variables utilizadas en el programa de ordenación.*

En la Figura 8.36 puede verse un organigrama del algoritmo a implementar. Al comienzo de la rutina se incluye un conjunto de instrucciones que sirven para especificar los parámetros iniciales del programa. En este caso se introducen las constantes 0 y 1, y se inicializan las variables *dT* (leyéndola de la posición 0010 de memoria) y *pfT* (sumando a la posición inicial de la tabla su tamaño). A continuación se entra en el bucle principal del programa, que se ejecutará *nT – 1* veces, una por cada vez que se obtiene un nuevo máximo de la zona no ordenada. El bucle interior recorre con el puntero *pF* la zona no ordenada de la tabla; es decir, de la posición *dT* a la *pfT*, dejando el nuevo máximo en esta última posición (*pfT*).

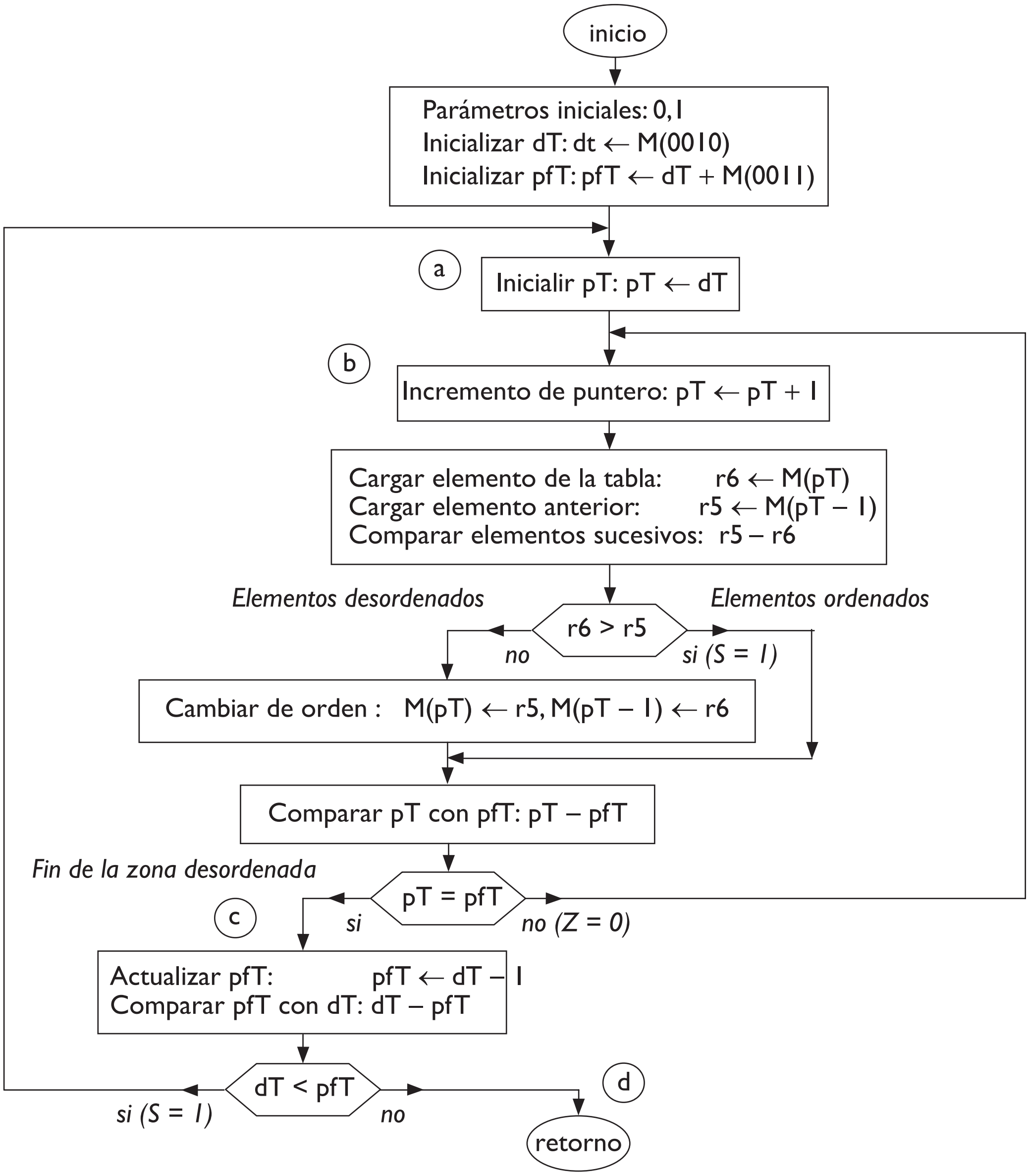


Figura 8.36. *Organigrama del programa de ordenación.*

Como el programa se configura como una rutina, finaliza con una instrucción RET. De acuerdo con el enunciado del problema, el programa que llame a esta rutina debe ubicar en la posición H’0010 de memoria la dirección de comienzo de la tabla, y en *H’0011* su tamaño. Una vez realizado lo anterior se llamará a la rutina con las dos siguientes instrucciones: *LLI rD,A0* y *CALLR*.

Asignación de registros y de memoria

En la Tabla 8.14 se incluye la asignación de registros y de posiciones de memoria para los parámetros, variables y el código máquina del programa.

Tabla 8.14. *Asignación de registros y de memoria del programa de ordenación.*

Parámetro o variable	Registro	Posición de memoria	Comentario
0000	r0		Para pasar el valor de un registro a otro
0001	r1		Para incrementar el índice i
dT	r2		Dirección de inicio de la tabla
pfT	r3		Dirección del final de la zona no ordenada
pT	r4		Puntero que recorre la zona no ordenda
M(pT – 1)	r5		Variable que contiene un elemento de la tabla
M(pT)	r6		Variable que contiene el siguiente elemento de la tabla
Programa		00A0	Dirección de carga del programa

Redacción del programa en nemónicos

Teniendo en cuenta el repertorio de instrucciones de CODE-2 (Tabla 8.5) y la asignación de memoria, el programa en nemónicos es el que se incluye en la tercera columna de la Tabla 8.15. La primera columna especifica simbólicamente las referencias de salto, de acuerdo con la notación utilizada en el organigrama y la segunda columna las direcciones de memoria en que se ubicaría cada instrucción. También en la quinta columna se incluye un breve comentario sobre la instrucción.

Tabla 8.15. *Programa de ordenación por el método de las burbujas.*

Rfcía. salto	Dircc. Memoria	Instrucción máquina		Explicación
		Nemónico	Hex	
	00A0	LLI r0, 00	2000	Inicializar r0 a 0
	00A1	LLI r1, 01	2101	Inicializar r1 a 1
	00A2	LLI rD, 10	2D10	Cargar dirección de inicio de tabla en rD
	00A3	LD r2, [00]	0200	Cargar dT en r2
	00A4	LD r3, [01]	0301	Cargar tamaño de tabla en r3
	00A5	ADDS r3, r3, r2	6332	Obtener dirección final de tabla (pfT)

Tabla 8.15. *Programa de ordenación por el método de las burbujas. (Cont.)*

Rfcía. salto	Dircc. Memoria	Instrucción máquina		Explicación
		Nemónico	Hex	
(a)	00A6	ADDS r4, r2, r0	6420	Inicializar pT para la búsqueda de un máximo
(b)	00A7	ADDS r3, r3, r1	6331	Incrementar pT
	00A8	ADDS rD, r4, r0	6D40	Llevar pT a registro de dirección
	00A9	LD r6, [00]	0600	Cargar elemento pT de tabla en r6
	00AA	SUBS rD, rD, r1	7DD1	Decrementar la dirección
	00AB	LD r5, [00]	0500	Cargar elemento pT-1 de tabla en r5
	00AC	LLI rD, B3	2DA7	Llevar a rD dirección (e) de salto
	00AD	SUBS rF, r5, r6	7F56	Comparar elementos pT – 1 y pT de la tabla
	00AE	BS	C200	Saltar a (e) si están ordenados
	00AF	ADDS rD, r4, r0	6D40	Llevar pT a registro de dirección
	00B0	ST [00], r5	1500	Cambio de orden en la tabla
	00B1	SUBS rD, rD, r1	7DD1	Obtener pT – 1 en rD
	00B2	ST [00], r6	1600	Cambio de orden en la tabla
(e)	00B3	SUBS rF, r3, r4	7F34	Comparar pfT con pT
	00B4	LLI rD, B8	2DB8	Cargar en rD dirección (c) de salto
	00B5	BZ	C100	Saltar a (c) si pft = pT
	00B6	LLI rD,A7	2DA7	Cargar en rD dirección (b)
	00B7	BR	C000	Saltar a (b)
(c)	00B8	SUBS r3, r3, r1	7331	Actualizar pfT
	00B9	SUBS rF, r2, r3	7F23	Comparar dT con pfT
	00BA	LLI rD, A6	2DA6	Cargar en rD dirección (a) para salto
	00BB	BS	C200	Saltar a (a) si pfT > dT
(d)	00BC	RET	E000	Retorno al programa de llamada

Codificación del programa en código máquina

La cuarta columna de la Tabla 8.15 contiene el programa codificado en código máquina (hexadecimal). Conviene indicar que el programa en lenguaje máquina está constituido tan sólo por la segunda columna (dirección de memoria) y la cuarta columna (código máquina) de la tabla indicada. Las otras columnas sirven como ayuda para facilitar la redacción del programa y mejorar su legibilidad

Instrucciones al operador

En la Tabla 8.16 se incluyen los comentarios para el usuario del programa.

Tabla 8.16. *Instrucciones para el uso del programa de ordenación.*

Rutina de ordenación de los elementos de una tabla	
Posición inicial de la rutina	**00A0**
La rutina ordena ascendentemente los datos que se encuentran a partir de la posición que se especifica en la dirección H'0010 de memoria. El número de datos a ordenar se encuentra en la dirección de memoria H'00A0. La rutina utiliza los registros r0 a r6, rD y rE.	

Consideramos que con los tres ejemplos de programas vistos en esta sección se tiene una visión clara de cómo se programa CODE-2 o cómo actúa en el nivel de lenguaje máquina. Al final del capítulo se incluyen enunciados de otros ejemplos de programas.

8.3.6 Utilización de CODE-2

En esta sección se analiza cómo usar CODE-2, concretamente se describe su panel frontal, y se explica cómo introducir los programas, cómo lanzar a ejecución un programa, cómo poder ver el contenido de una posición de memoria o de un registro, etc.

En la Figura 8.37 se muestra un esquema del panel frontal de CODE-2. Sus elementos principales son:

- Interruptores ON/OFF para conexión al suministro de energía eléctrica y reiniciación del sistema, y de PASO A PASO. Al conmutar este último interruptor, CODE-2 funciona en **Modo Paso a Paso**, de forma que entra en estado de espera (Figura 8.31) después de la ejecución de cada instrucción, pudiéndose ejecutar las siguientes instrucciones pulsando sucesivamente la tecla CONTINUAR. Para salir del Modo Paso a Paso hay que volver a conmutar el interruptor PASO A PASO.
- Teclado hexadecimal, para introducción de programas y datos. Este teclado actúa como puerto de entrada 1 (*IP1*), y tiene asociado en su parte superior un visualizador de 4 indicadores de 7 segmentos que siempre muestra el valor del puerto citado.
- Puertos de salida 1 (*OP1*) y 2 (*OP2*). La información se visualiza en hexadecimal, de forma que como cada puerto es de 16 bits se utilizan 4 indicadores por cada uno de los puertos.
- Teclas de órdenes:
  - Dirección.
  - Registros.
  - Cargar.
  - Ejecutar.
  - Continuar.
- Pilotos indicando el valor de los biestables indicadores de la *ALU* (*Z, S, C* y *V*) y visualizadores de los contenidos de los registros internos *IR* (registro de instrucción) y *PC* (Contador de Programa), cada uno formado por 4 indicadores de 7 segmentos. La información se visualiza en hexadecimal. Estos pilotos y visualizadores permiten la observación del funcionamiento interno de CODE-2, cuando éste funciona en Modo Paso a Paso.

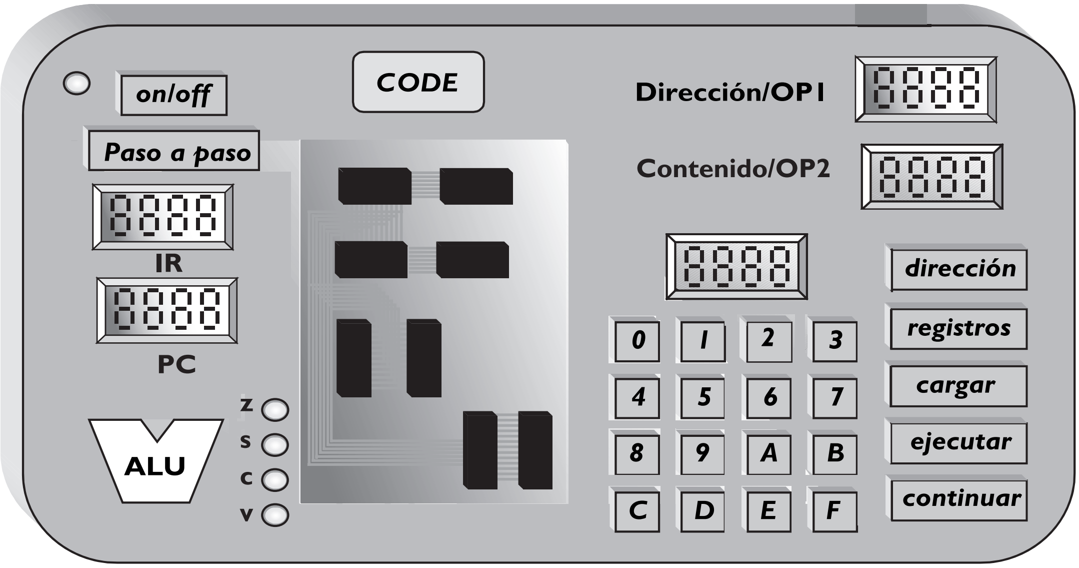


Figura 8.37. *Aspecto del panel frontal de CODE-2.*

Una vez activado el interruptor ON/OFF, bajo el control de las teclas de órdenes, pueden realizarse las siguientes tareas:

- **Seleccionar una posición de memoria.** Para ello en el teclado hexadecimal se teclea la dirección requerida y se pulsa la tecla DIRECCIÓN. Entonces en el puerto de salida *OP1* aparece la dirección tecleada y en el *OP2* el contenido de esa posición. Después de haber hecho la selección, si se pulsa sucesivamente CONTINUAR van apareciendo en *OP1* y *OP2* las direcciones y contenidos, respectivamente, de las siguientes posiciones de memoria. En otras palabras, en este contexto CONTINUAR hace que avance en una unidad la dirección seleccionada
- **Cargar una información en una posición de memoria.** Después de haber seleccionado una dirección, en el teclado hexadecimal se introduce la información a memorizar y se pulsa la tecla CARGAR. Al realizar esta acción además de memorizarse la información tecleada, se avanza en una unidad la dirección de memoria. Por consiguiente, puede cargarse en memoria un programa o una tabla de datos (por ejemplo), seleccionando la dirección inicial de la misma, y posteriormente tecleando la primera palabra a almacenar y pulsando CARGAR, la segunda palabra a almacenar y pulsando nuevamente CARGAR, y así sucesivamente.
- **Seleccionar los registros.** Pulsando la tecla REGISTROS, aparece en *OP1* el código del registro 0; es decir *0000*, y en *OP2* su contenido. Pulsando sucesivamente después CONTINUAR, van apareciendo en *OP2* el código del siguiente registro (*0001, 0002, ..., 000F*) y en *OP2* su contenido.
- **Cargar una información en un registro.** Después de haber seleccionado un registro, en el teclado hexadecimal se introduce la información a memorizar y se pulsa la tecla CARGAR. Al realizar esta acción, además de memorizarse la información tecleada en el registro seleccionado, se avanza en una unidad el código del registro.
- **Ejecutar un programa.** Primero se selecciona como dirección de memoria la dirección de la primera instrucción del programa, y después se pulsa EJECUTAR.

EJEMPLO 8.17

Para cargar y ejecutar el programa de suma de dos tablas descrito en la Sección 8.3.5 habría que realizar las operaciones que se indican en la Tabla 8.17.

Tabla 8.17. *Operaciones a realizar para cargar y ejecutar el programa de suma de dos tablas.*

Interruptor	Teclado hexadecimal	Tecla de órdenes	Explicación
ON/OFF			*Conexión eléctrica de CODE-2*
	0010	DIRECCIÓN	*Selección de la 1ª dirección del programa*
	2000	CARGAR	*Carga del programa (4ª columna de la Tabla 8.9.)*
	2101	CARGAR	
	2220	CARGAR	
	2C00	CARGAR	
	6DC0	CARGAR	
	0340	CARGAR	
	0480	CARGAR	
	15C0	CARGAR	
	6CC1	CARGAR	
	2DAF	CARGAR	
	7F2C	CARGAR	
	C200	CARGAR	
	2DA4	CARGAR	
	C000	CARGAR	
	F000	CARGAR	
	0010	EJECUTAR	*Ejecución del programa*
Paso a Paso			*Si se quiere ejecutar en modo paso a paso*
		CONTINUAR	*Se ejecuta la siguiente instrucción*
		CONTINUAR	*Se ejecuta la siguiente instrucción*
		CONTINUAR	*Se ejecuta la siguiente instrucción*
		CONTINUAR	*Se ejecuta la siguiente instrucción*
		CONTINUAR	*Se ejecuta la siguiente instrucción*
		CONTINUAR	*Se ejecuta la siguiente instrucción*
Paso a Paso			*Si se desea continuar en modo normal*

A continuación se da una idea de cómo se ha diseñado el sistema de interpretación de las teclas de órdenes. La memoria de CODE-2 se distribuye como indica la Figura 8.38; es decir, los usuarios no deben escribir desde la posición *F000* a la *FFFF*, aunque sí pueden leer esta zona. La parte ROM contiene programas monitores que gestionan las acciones de las teclas de órdenes. Concretamente, el interruptor on/off y las teclas DIRECCIÓN, REGISTROS y EJECUTAR provocan una interrupción que lanza a ejecutar una de las siguientes rutinas de servicio:

- Servicio de identificación de interrupción.
- Servicio de arranque.
- Servicio de acceso a memoria.
- Servicio de acceso a registros.

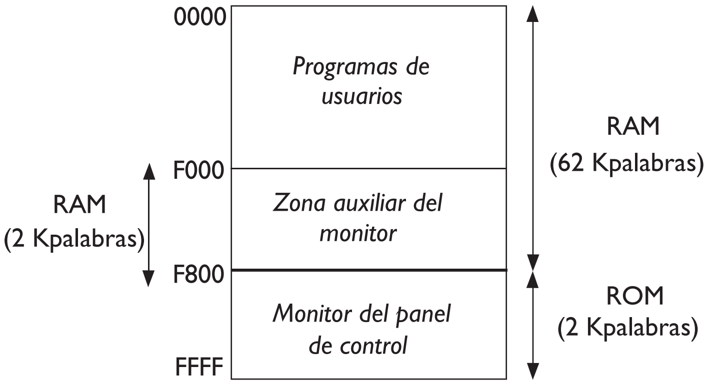


Figura 8.38. *Distribución de la memoria de CODE-2.*

Otra cuestión de interés para el programador es que las rutinas de servicio cambian el contenido del registro RF, por lo que si durante la ejecución de un programa se proyecta hacer uso de alguna de ellas, el programador debe prescindir de RF.

En el capítulo siguiente (Sección 9.6) se describe con más detalle cómo se ha implementado el sistema de control de las teclas de órdenes y las rutinas de servicio respectivas.

8.4 Descripción de un ordenador didáctico elemental a nivel de lenguaje ensambador

El lenguaje máquina tiene dos notables ventajas:

a) Es directamente interpretable y ejecutable por el procesador.
b) Los programas se adaptan completamente a los elementos y peculiaridades del computador ya que se programan directamente los elementos físicos de éste: registros, puertos, memoria, etc. Esto lleva consigo que se puedan desarrollar programas muy eficientes en cuanto a tiempo de ejecución y a consumo de capacidad de memoria.

No obstante (como veremos con más detalles en la Sección 8.6), las ventajas anteriores quedan obscurecidas, entre otros, por ciertos inconvenientes detectados claramente a lo largo de la sección anterior y que hacen al lenguaje máquina muy incómodo de utilizar:

a) Las instrucciones en lenguaje máquina se han de dar en binario, o en octal, o en hexadecimal (como ocurre con CODE-2). En cualquier caso un programa no es más que una serie

de caracteres numéricos, a partir de los cuales a simple vista no podemos concluir gran cosa sobre lo que realiza. En otras palabras, la redacción del programa es compleja y el programa resultante es muy poco *legible*.

b) El programador ha de realizar la **asignación de registros** y la **asignación de memoria**, decidiendo en qué posiciones colocar las diferentes unidades de que consta un programa (programa principal, subrutinas y cada uno de los datos).

Para solucionar estos dos inconvenientes, manteniendo casi todas las ventajas del lenguaje máquina, se utiliza el **lenguaje ensamblador**. En éste cada instrucción viene identificada mediante un nombre simbólico (nemónico) que, al igual que el código de operación en el lenguaje máquina, identifica las acciones a realizar por la instrucción. Además, en vez de tener necesariamente que especificar cada variable o cada constante indicando el registro o la posición de memoria donde se encuentra, se puede dar un nombre simbólico a cada una de ellas, reduciéndose así notablemente el problema de asignación de registros y de memoria para los datos. Para un usuario los programas resultan así mucho más fáciles de entender y desarrollar. No obstante lo anterior, un programa en lenguaje ensamblador no es directamente ejecutable por el procesador, requiriendo previamente una **traducción** de los nemónicos y nombres simbólicos a los códigos máquina y direcciones correspondientes. En los lenguajes ensambladores *cada* nombre simbólico de instrucción corresponde aproximadamente a *una* instrucción en código máquina, de manera que la traducción es muy sencilla. La traducción de un programa en lenguaje ensamblador a lenguaje máquina se realiza por medio de un programa traductor conocido como **ensamblador**; éste capta como datos de entrada las instrucciones del programa redactado en ensamblador y genera como resultado de salida el mismo programa pero escrito en lenguaje máquina. Hay que resaltar que el ensamblador no sólo cambia los nemónicos por códigos de operación, sino que además decide las direcciones de memoria y los registros concretos correspondientes a cada una de las variables o direcciones simbólicas utilizadas en el programa en ensamblador. Merced a estas características, junto a otras que se verán más adelante en esta misma sección y en la Sección 8.6, se facilita considerablemente la redacción y legibilidad de los programas con respecto al lenguaje máquina.

El proceso de traducción puede realizarse en el mismo computador que ejecutará el programa, o en un computador distinto. En este último caso el programa traductor se denomina **ensamblador cruzado**; éste es el caso de un ensamblador que hemos desarrollado para CODE-2 y que está ideado para entornos PC: el programa en ensamblador lo editamos y traducimos en un PC, y el programa máquina resultante lo cargamos en CODE-2 (o en un de emulador de éste) para su ejecución. Puede obtenerse el ensamblador de CODE-2 a través de la página web asociada a este libro, referencia.

Recordando el lenguaje máquina de CODE-2 (Tabla 8.5) una instrucción máquina se compone de:

- Código de operación, que identifica la instrucción.
- Operandos de la instrucción, que pueden ser de los siguientes tipos:
  - registros: *r0* a *rF*,
  - campo *v*: que representa un valor de 00 a FF, o
  - puertos de entrada (*IPv*) o de salida (*OPv)* que representan un valor de 00 a FF.

8.4.1 Formato de las instrucciones de ensamblador

El formato de las instrucciones en el lenguaje ensamblador que hemos ideado para CODE-2 es el siguiente:

[ETIQUETA:] NEMÓNICO OPERANDOS [;COMENTARIO]

donde cada campo debe ir separado al menos por un espacio en blanco, y.

Etiqueta

Es un campo alfanumérico opcional que se puede utilizar para identificar la posición de las instrucciones, y que debe ir al comienzo de la línea y acabar con el carácter "dos puntos" (:). En las instrucciones de salto se dará como operando el nombre de la etiqueta de la instrucción a la que se desea saltar, en lugar de tener que cargar previamente en el registro *rD* la dirección física de memoria donde se encuentra dicha instrucción. El ensamblador se encarga de asociar a cada etiqueta la dirección física de memoria donde se encuentra la instrucción correspondiente para generar adecuadamente las instrucciones de salto.

Nemónico

Es el nombre simbólico (nemónico) dado para cada instrucción en la segunda columna de la Tabla 8.18. El ensamblador consultará automáticamente una tabla interna para sustituir los nemónicos por los códigos de operación binarios correspondientes (columna primera de la Tabla 8.14. Obsérvese que los nemónicos del lenguaje ensamblador coinciden con los que hemos utilizados para las instrucciones del lenguaje máquina.

Tabla 8.18. *Instrucciones del lenguaje ensamblador.*

Nombre	Ensamblador CODE-2	Ejemplo	Significado del ejemplo	Ensamblador IEEE694
Cargar	LD rx, [rD + v]	LD r4, [01]	r4 ← M(rD + H'01)	LD .rx, /[.rE]v
Almacenar	ST [rD + v], rx	ST [00], rE	M(rD + H'00) ← rE	ST /[.rE]v, rx
Carga inmediata baja	LLI rx, v	LLI r7, 07	r7(15:8) ← H'00; r7(7:0) ← H'07	LLI .rx, #v
Carga inmediata alta	LHI rx, v	LHI r7, AB	r7(15:8) ← H'AB	LHI .rx, #v
Entrada	IN rx, IPv	IN rD, IP1	rD ← IP1	IN .rx, /v
Salida	OUT OPv, rx	OUT OPD'13	OP0D ← rx	OUT /v, .rx
Suma	ADDS rx, rs, ra	ADDS rD, r3, r4	rD ← r3 + r4	ADDS .rd, .rs, .ra
Resta	SUBS rx, rs, ra	SUBS rF, r1, r0	rF ← r1 – r0	SUBS .rd, .rs, .ra
NAND	NAND rx, rs, ra	NAND r7, r3, r5	r7 ← (r3 · r5)'	NAND .rd, .rs, .ra
Desplaza izquierda	SHL rx	SHL r6	C ← r6(15), r6(i) ← r6(i – 1), i = 15, …, 1; r6(0) ← 0	SHL .rx
Desplaza derecha	SHR rx	SHR r5	C ← r5(0), r5(i) ← r5(i + 1), i = 0, …, 14; r5(15) ← 0	SHR .rx
Desplaza arit. dcha.	SHRA rx	SHRA r8	C ← r8(0), r8(i) ← r8(i + 1), i = 0, …, 14	SHRA .rx
Salto incondicional	BR	BR	PC ← rD	BR [.rD]
Salto si resultado cero	BZ	BZ	Si Z = 1 , PC ← rD	BZ[.rD]
Salto si resultado negativo	BS	BS	Si S = 1 , PC ← rD	BS [.rD]
Salto si resultado con acarreo	BC	BC	Si C = 1 , PC ← rD	BC [.rD]

Tabla 8.18. *Instrucciones del lenguaje ensamblador. (Cont.)*

Nombre	Ensamblador CODE-2	Ejemplo	Significado del ejemplo	Ensamblador IEEE694
Salto si resultado con desbordamiento	BV	BV	Si V = 1 , PC ← rD	BV [.rD]
Llamada incondicional a subrutina	CALLR	CALLR	rE ← rE – 1, M(rE) ← PC, PC ← rD	CALLR [.rE]
Llamada si resultado cero	CALLZ	CALLZ	Si Z = 1, rE ← rE – 1, M(rE) ← PC, PC ← rD	CALLZ [.rE]
Llamada si resultado negativo	CALLS	CALLS	Si S = 1, rE ← rE – 1, M(rE) ← PC, PC ← rD	CALLS [.rE]
Llamada si resultado con acarreo	CALLC	CALLC	Si C = 1, rE ← rE – 1, M(rE) ← PC, PC ← rD	CALLC [.rE]
Llamada si resultado con desbordamiento	CALLV	CALLV	Si V = 1, rE ← rE – 1, M(rE) ← PC, PC ← rD	CALLV [.rE]
Retorno	RET		PC ← M(rE); rE ← rE + 1	RET
Parar	HALT		Parar	HALT

Observaciones:
- *sx, rs, ra* es un registro (r0 a rF o R0 a RF) o un nombre simbólico de registro (definido con EQU) o una variable-registro (definida con DR).
- *v* es un número de 0 a 255, en base decimal (D'número), hexadecimal (H'número o número), octal (Q'número) o binaria (B'número).

Operandos

Es un campo donde se incluyen los parámetros asociados a la instrucción: código de registros involucrados, valor *v*, o puerto de entrada o salida. Como se detallará más adelante, los operandos de cada instrucción se pueden dar:

- Especificando directamente el elemento o valor:
  - Registros: r0, r1, ..., rE, rF. El ensamblador no es sensible a mayúsculas y minúsculas; es decir, se puede usar también la notación: R0, R1, ..., RE, RF.
  - Valor *v* y puertos de entrada o salida: un número de 0 a 255 (8 bits); especificado según la normalización IEEE 694 (véase Sección 8.1); por ejemplo, D'193 ó Q'301 ó H'C1 ó B'11000001. Si no se especifica la letra identificadora de la base se interpretará que es la hexadecimal.
  - Código del puerto de entrada o salida: IPv ó Opv, respectivamente, donde *v* es un número de 0 a 255 (8 bits); especificado según la normalización IEEE 694 (véase Sección 8.1); por ejemplo, IPC1, ó IPH'C1 ó IPD'193 ó IPQ'301 ó IPB'11000001. Si no se especifica la letra identificadora de la base se interpretará que es la hexadecimal.
- Utilizando un nombre simbólico, en la forma que se indicará posteriormente.

Comentario

Es un campo alfanumérico opcional para incluir explicaciones o comentarios que hagan más legible el programa, y que debe ir precedido por el carácter "punto y coma" (;). Puede haber líneas del programa que sólo contengan comentarios, siempre precedidos por "punto y coma".

8.4.2 Directivas o pseudoinstrucciones

Además de las instrucciones correspondientes a los códigos máquina (Tabla 8.14), en el lenguaje ensamblador se utilizan cinco **directivas** o **pseudoinstrucciones**, que sirven para especificar la dirección inicial del programa, asignar nombres simbólicos a los parámetros de las instrucciones, dar valores iniciales a las variables, y poder insertar archivos en ensamblador previamente desarrollados. Las directivas tienen un formato similar al de las instrucciones, como se ve a continuación.

ORG

La **directiva ORG** (de "origen") se utiliza para indicar al ensamblador la dirección de memoria a partir de la que se ha de situar el segmento de programa que vaya a continuación. En este caso el campo *ETIQUETA* puede utilizarse para referenciar o identificar el segmento de programa que viene a continuación, y *OPERANDOS* es una dirección de memoria absoluta dada en base decimal, octal, hexadecimal o binaria. Hay que tener en cuenta que los usuarios de CODE-2 no pueden utilizar la memoria a partir de la posición H'F000 (véase Sección 7.xx).

EJEMPLO 8.18

La línea de ensamblador:

```
Media   ORG   H'BC72 ; inicio del programa de calcular los valores medios
```

EQU

La **directiva EQU** (de "equiparar") permite dar nombres alternativos o simbólicos a registros, puertos de entrada/salida y a valores *v* a incluir en las instrucciones máquina. En este caso el campo *ETIQUETA* se usa para dar el nombre simbólico, y el campo *OPERANDOS* para especificar el elemento original al que se desea renombrar.

EJEMPLO 8.19

Las siguientes líneas de ensamblador:

```
rtro_2     EQU   r2 ; a partir de ahora se puede usar como operador rtro_2 o r2
rtro_3     EQU   R3 ;a partir de ahora se puede usar como operador rtro_3 o R3
SP         EQU   rE ; a partir de ahora se puede usar SP en vez de rE
dirección  EQU   r2 ; a partir de ahora se puede usar dirección en vez de rD
teclado    EQU   H'01; puedo usar "IN rtro_3,teclado", en vez de "IN r3,H'01"
N          EQU   D'13; es lo mismo (a partir de ahora) usar N que 13
```

significan lo que se indica en el comentario correspondiente.

DW

La **directiva DW** (de *Define Word*) define una variable en memoria (que se indica en el campo *ETIQUETA*) asignándole una posición de memoria y dándole el valor que se especifique en el

campo *OPERANDOS*. Debido a que las palabras de memoria son de 16 bits el valor debe estar comprendido entre H’0000 y H’FFFF. En decimal se pueden dar números negativos y positivos (comprendidos entre –32.767 y –32.768). Recordar que los datos se almacenan en complemento a dos, con lo que si el valor de un número negativo y se da en hexadecimal, octal o binario debe ser codificado correctamente.

EJEMPLO 8.20

Las siguientes líneas de ensamblador:

debe	***DW***	***D’466***	***; en la memoria se define la variable “debe” con valor 466***
haber	***DW***	***D’-3756***	***; en la memoria se define la variable haber con valor –3.756***
Cliente	***DW***	***H’C732***	***; en la memoria se define la variable “cliente” con valor H’C732***

significan lo que se indica en el comentario correspondiente.

El ensamblador automáticamente asigna una dirección de memoria a cada variable definida con la directiva *DW*. En las instrucciones de carga (*LD*) y almacenamiento (*S*T), se pueden utilizar como parámetros las variables definidas con la directiva DW.

DR

La **directiva DR** (de *Define Register*) define una variable (que se especifica en el campo *ETIQUETA*) en un registro, dándole el valor que se especifique en el campo *OPERANDOS*. Debido a que los registros son de 16 bits, el valor debe estar comprendido entre H’0000 y H’FFFF. En decimal se pueden dar números negativos y positivos (comprendidos entre –32.767 y –32.768). Recordar que los datos se almacenan en complemento a dos, con lo que si el valor de un número es negativo y se da en hexadecimal, octal o binario ha de ser codificado correctamente.

EJEMPLO 8.21

Las siguientes líneas de ensamblador:

puntero	***DR***	***D’1024***	***; se reserva un registro para la variable “puntero” con valor inicial de 1.024***
n_itera	***DR***	***1***	***; se reserva un registro para la ;variable “n_itera”, y con valor inicial 1***
0	***DR***	***0***	***; en la memoria se define la variable “0” con valor 0***

significan lo que se indica en el comentario correspondiente.

En las instrucciones donde se utilizan registros como operandos, en lugar de éstos pueden utilizarse las **variables-registro** definidas con la directiva *DR*.

La directiva *DR* presenta las siguientes restricciones:

a) El ensamblador a las variables-registros les va asignando registros ordenadamente desde el *r0* en adelante. Así, en el ejemplo anterior asignaría el registro *r0* a la variable “puntero”, el registro *r1* a la variable “n_itera”, y el registro *r2* a la variable “0”. Obviamente

el usuario sólo podrá hacer referencia expresa a los registros no usados para las variables-registros; es decir, si definimos 5 variables registros, sólo podremos referenciar explícitamente a los registros r5 a rF, ya que los registros del r0 al r4 serán ocupados por las variables definidas con directivas *DR*.

b) Se pueden utilizar como máximo 13 variables-registros, a las que se les asignarán los registros del *r0* al *rC*. Esto es debido a que el registro *rD* debe reservarse para los direccionamientos indirectos y para almacenar las direcciones de saltos y llamadas a subrutinas, el registro *rE* se usa como puntero de pila, y el *rF* es utilizado por el monitor, y su valor puede no conservarse entre instrucciones.

INCLUDE

Esta directiva permite incluir, en cualquier punto del programa ensamblador, un archivo también en ensamblador. El nombre del archivo se especifica entre comillas en el campo *OPERANDOS.* Por ejemplo, la línea:

INCLUDE "LIB_TABLAS.ASM"

Inserta, en el lugar donde se encuentra, las instrucciones en ensamblador que contiene el archivo LIB_TABLAS.ASM.

END

Esta directiva indica el final del programa en ensamblador.

Pseudoinstrucciones para direcciones de saltos

El programador de ensamblador no tiene necesidad de establecer las direcciones de los datos ni las direcciones de los saltos para las instrucciones de salto (*B-*) y de llamadas a subrutinas (*CALL-*). Recuérdese que las instrucciones a las que hay que saltar deben necesariamente llevar una etiqueta; puede ser por ejemplo, *lazo*. Para indicar al traductor de ensamblador la dirección de salto deben utilizarse las instrucciones siguientes:

LLI rD, LO(lazo)

LHI rD, HI(lazo)

donde *lazo* representa a la etiqueta de la instrucción a donde hay que saltar o donde comienza la subrutina.

El traductor de ensamblador asocia a la etiqueta *lazo* la dirección donde se encuentra, y al traducir las instrucciones citadas anteriormente genera las instrucciones máquina adecuadas para cargar en *rD* la dirección correspondiente a la etiqueta *lazo*.

8.4.3 Ejemplos de instrucciones en ensamblador

Una vez descritos el formato de las instrucciones en ensamblador y sus directivas, resulta conveniente examinar algunos ejemplo de cada una de las instrucciones (Tabla 8.14).

LD (cargar un registro con un dato de la memoria)

La instrucción de carga admite los formatos que se indican a continuación con ejemplos:

LD r3,[rD + H'5C] ; formato básico: se explicitan rD y el valor de *v.*

LD r3,[H'5C] ; *siempre* se suma al valor de *v* el contenido de rD, por lo que no es necesario explicitar *rD.*

LD r3,[rD] ; el ensamblador ahora supone que *v = H'00.*

LD r3,[N] *; N* debe haber sido definido previamente con una directiva EQU, y es el valor que se le suma a rD para obtener la dirección de memoria del dato.

LD r3,debe ; "debe" es una palabra en memoria que debe haberse definido con la directiva *DW.* El ensamblador genera las instrucciones máquina correspondientes: cargando en *rD* la dirección que tiene asignada la variable "debe" y después la instrucción LD r3,[rD + H'00].

LD saldo,debe ; "saldo" es un registro establecido con una directiva EQU o como variable-registro. El ensamblador genera las instrucciones correspondiente para llevar el valor de "debe" que está en la memoria principal, al registro asignado a la variable "saldo".

Obsérvese que cuando el dato se especifica por medio de una variable definida en memoria (como en los dos últimos ejemplos), el ensamblador debe traducir la instrucción *LD* a dos o tres instrucciones máquina: una o dos para cargar en el registro *rD* la dirección de la variable definida en memoria, y otra que será la propia instrucción de carga. Esto deber ser tenido muy en cuenta por el programador de ensamblador ya que en este caso el valor de *rD* previo a la instrucción de carga no se conserva.

ST (almacenar el contenido de un registro en la memoria)

Los formatos que admite esta instrucción son análogos a la instrucción anterior (carga de un registro). No obstante, hay una singularidad que se pone de manifiesto con el siguiente ejemplo:

ST media,r3

En este ejemplo, si la variable "media" no ha sido definida con una directiva *DW,* el ensamblador crea automáticamente esa variable en memoria. En otras palabras, si al escribir un programa en ensamblador no hay un número suficiente de registros para almacenar las variables, éstas pueden llevarse directamente a memoria, sin necesidad de definirlas previamente con directivas *DW.*

Cuando la referencia a memoria se realiza con una variable y no con una dirección, al igual que ocurre con la instrucción *LD,* no se conserva el valor previo de *rD.*

LLI (carga inmediata baja)

La instrucción de carga inmediata del byte menos significativo de un registro admite los formatos que se indican a continuación con ejemplos:

LLI r4,H'12 ; formato básico.

LLI R4,H'12 ; el ensamblador no es sensible a mayúculas/minúsculas.

LLI rtro_3,N ; "rtro_3" debe haber sido definido previamente con una directiva EQU o DR, y "N" con una directiva EQU.

LHI (carga inmediata alta)

Los formatos que admite esta instrucción son análogos a la instrucción anterior (carga inmediata baja).

IN (entrada)

La instrucción de entrada (llevar el contenido de un puerto de entrada a un registro) admite los formatos que se indican a continuación con ejemplos:

IN r4,IP13 ; formato básico: Se lleva al registro r4 el contenido del puerto de entrada H'13.

IN r4,IPD'19 ; se lleva al registro r4 el contenido del puerto de entrada D'19.

IN x,teclado ; la variable "x" debe haber sido definida con una EQU o como variable-registro; y "teclado" con una directiva EQU.

OUT (salida)

Los formatos que admite esta instrucción son similares a la instrucción anterior (entrada):

OUT OP03,rA	; formato básico.
OUT visualizador, y	; la variable "y" debe haber sido definida con una EQU o como variable-registro; y "visualizador" con una directiva EQU.

ADDS (suma), SUBS (resta), NAND (operación lógica NAND), SHL (desplazamiento a izquierda), SHR (desplazamiento a derecha), y SHRA (desplazamiento aritmético a derecha)

Todas las instrucciones arirmético-lógicas hacen referencia a registros, y estos pueden indicarse explícitamente o por medio de variables-registros o con nombres signados con directivas EQU. Algunos ejemplos son:

ADDS rD,rD,r1	; formato básico.
ADDS dirección,dirección,1	; los registros "dirección" y "1" deben haber sido definidos previamente con directivas EQU o DR.
SUBS SP,SP,1	; los registros "SP" y "1" deben haber sido definidos previamente con directivas EQU o DR.

B- (salto)

La instrucción de salto admite los formatos que se indican a continuación, con ejemplos:

LLI rD, LO(paso2)	
LHI rD, HI(paso2)	
BR paso2	; salto incondicional a la instrucción cuya etiqueta es *paso2*.
LLI rD, LO(a1)	
LHI rD, HI(a1)	
BZ	; salto si Z = 1 a la instrucción cuya etiqueta es *a1*.
LLI rD, LO(beta)	
LHI rD, HI(beta)	
BS	; salto si S = 1 a la instrucción cuya etiqueta es *beta.*
LLI rD, LO(etiqueta_3)	
LHI rD, HI(etiqueta_3)	
BC	; salto si C = 1 a la instrucción cuya etiqueta es *etiqueta_3.*
LLI rD, LO(e7)	
LHI rD, HI(e7)	
BV	; salto si V=1 a la instrucción cuya etiqueta es *e7.*

CALL- (llamada a subrutina)

Los formatos que admite esta instrucción son similares a la instrucción anterior (salto).

RET (retorno de subrutina) y HALT (parada)

Estas instrucciones no contienen ningún operando.

8.4.4 Ambigüedades

El ensamblador utiliza nombres simbólicos definidos por el programador como etiquetas, o con la instrucción de almacenamiento (ST) o con las directivas EQU, DW y DR. El ensamblador, para generar adecuadamente el código máquina, se encarga de asociar a cada nombre simbólico

una dirección de memoria o un registro, dependiendo del caso. Es extremadamente importante que el programador no genere ambigüedades, tales como:

- Utilizar la misma etiqueta para más de una instrucción.
- Utilizar como nombre de una variable-registro el nombre original de un registro (r1 a rF o R1 a RF).
- Utilizar un mismo nombre simbólico para dos variables distintas.

El traductor de ensamblador trata de detectar los posibles errores de programación.

8.4.5 Ejemplo de programa en ensamblador

A continuación damos un ejemplo completo de un programa escrito en ensamblador.

EJEMPLO 8.22

A partir de la posición A730 de la memoria de CODE-2 se tiene una cadena de caracteres Unicode, que finaliza con un carácter CR ("retorno de carro"). Hacer un programa que sustituya en la cadena los caracteres punto y coma (;) por coma (,) y proporcione por el puerto de salida OP2 el número de sustituciones que se efectúen.

Parámetros y variables

Vamos a utilizar los siguientes parámetros y variables:

- ***CR****, carácter de control CR. Teniendo en cuenta que Unicode incluye los caracteres ISO Latín 1, con el primer byte 0, el código de CR resulta ser H'000D (véase Apéndice).*
- ***PYC****, carácter punto y coma (;). Utilizando el mismo argumento del carácter CR, resulta tener el código H'003B.*
- ***C****, carácter coma (,). Su código es H'002C.*
- ***UNO****, constante 1.*
- ***P****, puntero de la tabla. Su valor inicial debe ser H'A730.*
- ***EC,*** *carácter a analizar de la cadena.*
- ***NC****, número de cambios. Esta variable debe inicializarse a 0.*

Descripción del programa

La descripción del algoritmo en seudocódigo es la siguiente:

Paso 1) Captar un elemento de la cadena; es decir hacer: EC ← M(P)

Paso 2) Comparar con CR; es decir hacer: EC-CR

Paso 3) Si Z = 1 ir al Paso 11, sino continuar

Paso 4) Comparar con PYC; es decir hacer: EC-PYC

Paso 5) Si Z = 1 ir al Paso 8, sino continuar

Paso 6) Incrementar en uno el puntero de la tabla: P ← P + 1

Paso 7) Ir al Paso 1.

Paso 8) Cambiar el carácter de la cadena; es decir, hacer M(P) ← C

Paso 9) Incrementar la variable NC; es decir, hacer: NC ← NC + 1

Paso 10) Ir al Paso 6

Paso 11) Proporcionar por el puerto de salida OP2 el valor de NC

Paso 12) Fin del programa

EJEMPLO 8.22 (Cont.)

Programa

El programa en ensamblador es el siguiente:

; programa para sustituir ";" por "," en una cadena de caracteres Unicode

```
ORG          DR     H'A000       ; el programa comienza en A000
      CR     DR     H'000D       ; carácter "CR"
      PYC    DR     H'003B       ; carácter ";"
      C      DR     H'002C       ; carácter ","
      uno    DR     H'1          ; constante 1
      P      DR     H'A730       ; puntero de la cadena
      NC     DR     H'0          ; número de sustituciones
      0      EQU    0            ; constante 0
      EC            r7           ; elemento de la cadena en análisis

      P1:    ADDS   rD,P,0       ; llevar a rD el puntero de la cadena
             LD     EC,[rD]      ; captar carácter de la cadena a analizar
             SUBS   rF,EC,CR     ; comparar con carácter "CR"
             LLI    rD,LO(P11)   ; Byte bajo de dirección de salto (P11)
             LHI    rD,HI(P11)   ; Byte alto de dirección de salto (P11)
             BZ                  ; saltar a Paso 11 si EC = "CR"
             SUBS   rF,EC,PYC    ; comprobar si EC es ";"
             LLI    rD,LO(P8)    ; Byte bajo de dirección de salto (P8)
             LHI    rD,HI(P8)    ; Byte alto de dirección de salto (P8)
             BZ                  ; si EC = ";" saltar a Paso 8
      P6:    ADDS   P,P,uno      ; actualizar el puntero de la cadena
             LLI    rD,LO(P1)    ; Byte bajo de dirección de salto (P1)
             LHI    rD,HI(P1)    ; Byte alto de dirección de salto (P1)
             BR                  ; salto incondicional a Paso 1
      P8:    ADDS   rD,P,0       ; llevar a rD el puntero de la cadena
             ST     [rD],C       ; sustituir en la cadena ";" por ","
             ADDS   NC,NC,uno    ; incrementar en 1 el número de cambios
             LLI    rD,LO(P6)    ; Byte bajo de dirección de salto (P6)

             LHI    rD,HI(P6)    ; Byte alto de dirección de salto (P6)
             BR                  ; salto incondicional a Paso 6
      P11:   OUT    OP02,NC      ; dar en salida el número de cambios
             HALT                ; fin de programa
END                              ; fin de programa en ensamblador
```

EJEMPLO 8.22 (Cont.)

Instrucciones de uso

El programa CPYC, cambia los punto y comas por comas en una cadena de caracteres que se inicia en la posición A730. El programa se almacena a partir de la posición A000 de la memoria, y proporciona por el puerto OP2 el número de cambios efectuados.

La ejecución del ensamblador en un compatible PC, utilizando como archivo de datos de entrada el programa anterior se genera el programa siguiente en lenguaje máquina (en la primera columna se da la dirección y en la segunda la instrucción máquina correspondiente):

A000	200D
A001	213B
A002	222C
A003	2301
A004	2430
A005	34A7
A006	2500
A007	2600
A008	6D46
A009	0700
A00A	7E70
A00B	2D1C
A00C	3DA0
A00D	C100
A00E	7F71
A00F	2D16
A010	3DA0
A011	C100
A012	6443
A013	2D08
A014	3DA0
A015	C000
A016	6D40
A017	1200
A018	6553
A019	2D12
A01A	3DA0
A01B	C000
A01C	5025
A01D	F000

EJEMPLO 8.22 (Cont.)

Para mejor entender lo que hacen las distintas instrucciones máquina de este programa, a continuación se vuelve a reproducir incluyendo los nemónicos y comentarios.

A000	200D	LLI	R0,0D	; carácter "CR"
A001	213B	LLI	r1,3B	; carácter ";"
A002	222C	LLI	r2,2C	; carácter ","
A003	2301	LLI	r3,01	; constante 1
A004	2430	LLI	r4,30	; puntero de la cadena
A005	34A7	LHI	r4,A7	
A006	2500	LLI	r5,00	; número de sustituciones
A007	2600	LLI	r6,00	; constante 0
A008 (P1)	6D46	ADDS	RD,r4,r6	; llevar a rD el puntero de la cadena
A009	0700	LD	r7,[00]	; captar carácter de la cadena a analizar
A00A	7E70	SUBS	rF,r7,r0	; comparar con carácter "CR"
A00B	2D1C	LLI	rD,1C	; dirección de P11
A00C	3DA0	LHI	rD,A0	
A00D	C100	BZ		; saltar a Paso 11 si EC = "CR"
A00E	7F71	SUBS	rF,r7,r1	; comprobar si EC es ";"
A00F	2D16	LLI	rD,16	; dirección de P8
A010	3DA0	LHI	rD,A0	
A011	C100	BZ		; si EC = ";" saltar a Paso 8
A012 (P6)	6443	ADDS	r4,r4,r3	; actualizar el puntero de la cadena
A013	2D08	LLI	rD,08	; dirección de P1
A014	3DA0	LHI	rD,A0	
A015	C000	BR		; salto incondicional a Paso 1
A016 (P8)	6D40	ADDS	rD,r4,0	; llevar a rD el puntero de la cadena
A017	1200	ST	[rD],r2	; sustituir en la cadena ";" por ","
A018	6553	ADDS	r5,r5,r3	; incrementar en 1 el número de cambios
A019	2D12	LLI	rD,12	; dirección de P6
A01A	3DA0	LHI	rD,A0	
A01B	C000	BR		; salto incondicional a Paso 6
A01C (P11)	5025	OUT	OP02,r5	; dar en salida el número de cambios
A01D	F000	HALT		; fin de programa

Obsérvese que, como en lenguaje máquina siempre que haya una referencia a una dirección de memoria hay que cargar previamente en rD el valor oportuno, las instrucciones de salto en ensamblador se han transformado en tres instrucciones máquina: LLI, LHI y BR (o BZ). También puede observarse cómo las inicializaciones de registro, realizadas en ensamblador con la directiva DR se transforman en instrucciones LLI, y LHI en su caso (son las 8 primeras instrucciones del programa).

8.5 Lenguajes ensambladores. Normalización IEEE 694

Una de las características más relevantes de los lenguajes máquina y ensambladores (como se analizará con detalle en la Sección 8.6) es su dependencia del procesador: dos modelos

de procesador distintos tienen distintos lenguajes máquina y ensamblador. No obstante, la terminología para la definición de los nemónicos, así como la forma y orden de indicar los operandos (y su direccionamiento) puede normalizarse notablemente, con la consiguientes ventajas para los usuarios de estos tipos de lenguajes.

La asociación IEEE ha propuesto una normalización para lenguajes ensambladores de microprocesadores (IEEE 694) [IEE85], que debido a su interés resumimos a continuación. Es necesario hacer hincapié que la normativa sólo pretende ser un marco terminológico, y que las posibilidades reales (instrucciones concretas, tipos de direccionamientos, etc.) dependerán del procesador concreto.

8.5.1 Direccionamientos y operandos

La forma de expresar en las instrucciones los distintos tipos de direccionamientos es la que se indicó en la Sección 8.1 (véase ejemplos en la Tabla 8.1).

Si no se especifica nada el tipo de operando coincide con la palabra máquina. Las constantes numéricas pueden expresarse en distintas bases, siendo la base por defecto la decimal. El convenio de notación de un número en una base determinada ya se dio en la Sección 8.1 (un número debe ir precedido de los caracteres B', D', H' o Q', dependiendo de que se de en base binaria, decimal, hexadecimal u octal, respectivamente).

8.5.2 Formatos de las instrucciones

Los distintos campos (etiqueta, nemónico y de operandos) deben ir separados por al menos un espacio en blanco.

Las **etiquetas** deben ir al comienzo de la línea, y *pueden* acabar con dos puntos (:); es decir, coinciden con las del ensamblador de CODE-2, visto en la sección anterior.

El orden de los **operandos** para las operaciones aritméticas es el orden que se indicó en la Tabla 8.2. Sin embargo, en las instrucciones de transferencia de datos se da en primer lugar la fuente y después el destino. Así, el significado de las siguientes instrucciones es el que se indica:

MOV.3,.7 significa $R7 \leftarrow R3$

Los comentarios se indican como vimos en la Sección 8.4; es decir, precedidos por ";".

Los **nemónicos** más importantes son los que se indican en la Tabla 8.19. En la Tabla 8.20 se incluyen los sufijos utilizables en algunas de las instrucciones.

Tabla 8.19. *Resumen de los nemónicos definidos en el estándar IEEE 694.*

Nemónico	Explicación	Sufijos
	Instrucciones de transferencia de datos	
LD	Cargar	
ST	Memorizar	
MOVE	Llevar de fuente a destino	
FILL	Memorizar en varias posiciones	
PUSH	Apilar el operando	

Tabla 8.19. *Resumen de los nemónicos definidos en el estándar IEEE 694. (Cont.)*

Nemónico	Explicación	Sufijos
	Instrucciones de transferencia de datos	
POP	Desapilar el operando	
XCH	Intercambiar operandos	
IN	Entrada desde puerto a registro o memoria	
OUT	Salida a puerto desde registro o memoria	
	Instrucciones aritméticas	
ADD-, SUB-	Suma o Resta	**-U,-S,-C,-R**
MUL-, DIV-	Multiplicación o División	**-U.-S**
TEST	Se ajustan los indicadores de acuerdo con el ope.	
TESTSET	Lee y actualiza una posición de memoria	
NEG	Cambia de signo al operando	
ADJ	Ajuste a BCD	
CVT	Cambia el operando de formato o tipo	
	Instrucciones lógicas y de desplazamiento	
AND,ORXOR	Producto o Suma lógica	
NOT	Or-exclusivo	
SET, CLR	Negación lógica	**-C,-V**
SHL, SHR	Pone el operando todo a unos o a ceros	**-C,-V**
SHLA,SHRA	Desplazamiento a izda. o derecha	
ROL, ROR	Desplazamiento aritmético (el signo se preserva)	
ROLC, RORC	Rotación a izda. o dcha. (sin acarreo)	
	Rotación a izda. a través de biestable de acarreo	
	Rotación a dcha. a través de biestable de acarreo	
	Instrucciones de transferencias de control	
-BR	Salto incondicional	**D-,I-**
-B-	Salto condicional	*subfijos para*
CALL-	Llamada a procedimiento	*B-, CALL-,RET-:*
RET-	Retorno de procedimiento	**-Z,-NZ,-E,-NE**
RETI	Retorno de rutina de interrupción	**-GT,-GE,-LT,-LE**
RETSKIP	Retorno a la dirección de llamada incrementada en operando	**-H,-NH,-L,-NL -C,-P,-N,-V,PE**
SKIP-	Salta hacía adelante una o varias instrucciones	**-NC,-NP,-NN,-NV,-PO,-T,-F**

Tabla 8.19. *Resumen de los nemónicos definidos en el estándar IEEE 694. (Cont.)*

Nemónico	Explicación	Sufijos
	INSTRUCCIONES MISCELANEAS	
BRK	Iniciar una secuencia de interrupción	
CC-	El oper. destino toma el valor lógico de la condic.	
EI, DI	Habilitar o inhabilitar interrupciones	
ENTER	Suministro de protocolos de entrada (**ENTER**)	
EXIT	y salida (**EXIT**) de procedimientos	
EXT	Extracción de campo de bits	
HALT	Parar la CPU	
INS	Inserción de campo de bits	
NOP	No se cambia nada, se avanza a la instr. siguiente	
TR	Reempzar. oper. direcc. de una tabla por contenido	
WAIT	Parar hasta que se produzca una interr. externa	

Tabla 8.20. *Sufijos utilizados en algunas instrucciones definidas por el estándar IEEE 694.*

Sufijos	Significado
Sufijos de instrucciones aritméticas	
-U	No se considera el signo
-S	Se considera el signo
-C	Suma de operandos con acarreo anterior
-R	Resta en orden inverso
Sufijos de instrucciones lógicas y de desplazamiento	
-C	Biestable de acarreo
-V	Biestable de desbordamiento
Sufijos de instrucciones de transferencia de control	
D-, I-	Decrementar o incrementar operando antes del salto
-Z, -NZ	Si cero o no cero
-E, -NE	Si igual o distinto
-GT, -GE	Si mayor o, mayor o igual
-LT, -LE	Si menor o, menor o igual
-H, -NH	Si mayor o no mayor
-L, -NL	Si menor o no menor
-C, -NC	Si acarreo 1 ó 0
-P, -NP	Si positivo o no positivo
-N. -NN	Si negativo o no negativo
-V, -NV	Si desbordamiento o no
-PE, -PO	Si paridad par o impar
-T, -F	Si “verdad” o “falso”

Algunas de las ***directivas*** contempladas en el estándar son:

- **ORG.** El operando indica la dirección a partir de la cual deben cargarse los datos e instrucciones que se dan a continuación.
- **RES.** Reserva el bloque de memoria cuya longitud se indica en el campo de operandos. La etiqueta simboliza la primera posición del bloque.
- **DATA.** Las posiciones sucesivas de memoria son inicializadas con los valores que se dan (separados por comas) en el campo de operandos. A continuación de *DATA* puede incluirse un sufijo *.m*, indicando m el tamaño o tipo de los datos:

  DATA H'7A,H'3B,H'51

  reserva tres posiciones de memoria, que inicializa con los valores hexadecimales 7A, 3B y 51.
- **ASSIGN.** Asigna al símbolo indicado en la etiqueta el valor dado en el campo de operando.
- **EQU.** Asocia al símbolo que se incluye como etiqueta una constante o una expresión aritmética. Por ejemplo:

  RX: EQU.2

  permite denominar en las instrucciones al registro.2 utilizando el símbolo RX.

  UNO: EQU 1

  permite utilizar como operando tanto 1 como UNO
- **EXPORT.** El símbolo indicado se exporta a otros módulos.
- **IMPORT.** El símbolo que se cita ha sido definido en otro módulo.
- **RADIX.** Cambia la base de numeración implícita en la que se dan las constantes.
- **HEADING.** Hace que al comienzo de las páginas del listado ensamblador se incluya el texto que se dé en el campo de operandos (entre comillas, "").
- **PAGE.** Provoca que el ensamblador avance el listado de ensamblador al tope de la página siguiente.
- **END.** Informa al traductor que ha acabado el módulo del programa. Puede incluir un operando indicando la posición a partir de la que hay que comenzar a ejecutar el programa; es decir, el valor inicial del contador de programa.

En la última columna de la Tabla 8.18 se dan las instrucciones de CODE-2 expresadas según la norma IEEE 694. No hemos utilizado en su totalidad esta norma para definir el ensamblador de CODE-2 por cuestiones didácticas, ya que, como se observa al comparar la segunda y última columnas de la tabla citada, el IEEE 694 da lugar a notaciones mucho más complejas por haber sido proyectado para procesadores con muy variados tipos de direccionamiento, no siendo éste el caso del Computador Didáctico Elemental.

8.6 Comparación entre lenguajes máquina y ensamblador

En la Sección 8.3.3 hemos visto con cierto detalle el *lenguaje máquina* (o *código máquina*) del computador CODE-2. Aunque CODE-2 es una máquina muy elemental, a partir de la experiencia en la utilización de su lenguaje máquina podemos extraer una serie de características de los lenguajes máquina, independientemente de la complejidad y potencia del computador a que pertenezcan.

La peculiaridad más sobresaliente es que el lenguaje máquina es muy engorroso de utilizar, necesitando el programador conocer la arquitectura física del computador con cierto detalle.

La estructura del lenguaje máquina está totalmente adaptada a los circuitos del computador y muy alejada de la forma o lenguaje habitual en el que solemos expresar y analizar los problemas hoy día resolubles con computadora. Por ejemplo, para hacer cálculos aritméticos disponemos de un lenguaje o simbología matemática fácil de comprender y clara, que no se parece en nada al código máquina necesario para hacer dichos cálculos. Frente a esto, un programa escrito en lenguaje máquina es directamente interpretable por el procesador central: una vez introducido el programa en el computador (en su memoria principal), no se necesitan transformaciones previas para ser ejecutado (como ocurre con programas escritos en lenguajes simbólicos). Además un programa en código máquina se ejecuta muy eficientemente (con rapidez), ya que el usuario lo redacta específicamente para los circuitos que lo han de interpretar y ejecutar, y desde el código máquina se puede utilizar la totalidad de los recursos de la máquina.

Las principales características de los lenguajes máquina son las siguientes:

a) *Las instrucciones son cadenas de ceros y unos*, pudiéndose dar estas cadenas al computador en un código intermedio (octal o hexadecimal, por ejemplo).

b) *Los datos se utilizan por medio de las direcciones de memoria* donde se encuentran. En las instrucciones no aparecen nombres de variables (tal como x, y, z, etc.), sino que el programador debe hacer una *asignación de direcciones de memoria* para todas las variables y constantes del programa. Esta asignación debe tener en consideración la zona de memoria que ocupa el programa, pues pueden producirse solapamientos indeseables.

c) *Las instrucciones realizan operaciones muy simples.* Muchos computadores, por ejemplo, no disponen de instrucciones específicas de multiplicar y dividir; el programador debe ingeniárselas para expresar cada una de las operaciones que desee realizar en términos de las elementales instrucciones del repertorio máquina. Si desea multiplicar y la multiplicación no forma parte del repertorio, debe codificar un algoritmo que realice la multiplicación por medio de sumas, comparaciones, desplazamientos, etc., aunque eventualmente estas funciones pueden estar incluidas como módulos de biblioteca.

d) *Existe muy poca versatilidad para la redacción de las instrucciones.* Éstas tienen un formato rígido en cuanto a posición de los distintos campos que configuran la instrucción (código de operación, direcciones de memoria, códigos de puertos, etc.). El código de operación debe seleccionarse estrictamente entre los que figuran en una tabla o repertorio fijo. Todas las sentencias del programa máquina, por otra parte, son instrucciones, no existiendo sentencias declarativas.

e) *El lenguaje máquina depende y está ligado íntimamente al procesador* del computador. Si dos computadores tienen procesadores diferentes (uno no es una réplica del otro), tienen distintos lenguajes máquina. En particular, dos microcomputadoras con el mismo microprocesador (Power PC, por ejemplo, en el caso de algunas microcomputadoras personales PC) tienen igual lenguaje máquina. La dependencia del lenguaje máquina de la configuración del procesador hace que los programas redactados en este lenguaje de programación no sean *transferibles* o *trasladables* de un computador a otro. Un programa cualquiera en código máquina de un IBM RS/6000 no puede ser ejecutado directamente por un DG Aviion.

f) *En un programa en código máquina no pueden incluirse comentarios* que faciliten la legibilidad del mismo. Además, debido a su representación totalmente numérica, es muy difícil de reconocer o interpretar por el usuario.

Las limitaciones señaladas para los lenguajes máquina se resuelven con los lenguajes simbólicos (lenguajes ensambladores y lenguajes de alto nivel). Estos lenguajes facilitan

notablemente el trabajo de programación y hacen los programas más legibles. A continuación analizaremos las características de los lenguajes ensambladores (en contraposición con los lenguajes máquina). En el Capítulo 10 se estudiarán los lenguajes de alto nivel.

Según se vio con el ensamblador de CODE-2 (Sección 8.4), este tipo de lenguajes permiten al programador:

a) Escribir las instrucciones utilizando para representar los códigos de operación una *notación simbólica o nemotécnica,* en vez de códigos numéricos (binario, octal, etc.). Normalmente los códigos nemotécnicos están constituidos por tres o cuatro letras que, en forma abreviada indican la operación a realizar. Usualmente, debido al origen anglosajón de los fabricantes de computadoras, los nemotécnicos son abreviaturas en inglés. La suma, por ejemplo, en la mayoría de los ensambladores se representa como "ADD".
b) Utilizar *direcciones simbólicas* de memoria, en lugar de direcciones binarias absolutas. Así, por ejemplo, los datos pueden ser referenciados como X, Y, Z, A, B, DEBE, HABER, CONT, etc. Existen sentencias declarativas (también denominadas *directivas* o *seudoinstrucciones*) para indicar al traductor la correspondencia entre direcciones simbólicas y direcciones de memoria. Con estas pseudoinstrucciones el traductor crea una tabla con cuya ayuda, al generar las instrucciones máquina, sustituye las direcciones simbólicas por las direcciones binarias correspondientes.
c) Insertar líneas de *comentarios* entre las líneas de instrucciones. El traductor las elimina automáticamente, no incluyéndolas en el código máquina que genera.

Este tipo de lenguajes hace corresponder a cada instrucción en ensamblador una instrucción en código máquina.

El traductor de lenguaje ensamblador a lenguaje máquina se denomina *ensamblador*, y mejora o resuelve los problemas a), b), y f), citados al referirnos a los lenguajes máquina, persistiendo las limitaciones c), d) y e). Un programa en ensamblador no puede ejecutarse directamente por la computadora, siendo necesario ser traducido (*ensamblado*) previamente (véase Capítulo 11).

Hay unos lenguajes, desarrollados a partir de los ensambladores, que se denominan *macroensambladores*. Con ellos se solventa en cierta medida la limitación de tener un repertorio de instrucciones muy reducido. Los lenguajes macroensambladores disponen de *macroinstrucciones*, como por ejemplo transferir un bloque de datos de memoria principal a disco, multiplicar, dividir, etc. La macroinstrucción es una llamada a un módulo o rutina de una biblioteca, llamada *macro*, que el traductor inserta, previamente a realizar el proceso de generación del código máquina definitivo, en el lugar de la llamada correspondiente. Obviamente a cada macroinstrucción, a diferencia de las instrucciones, le corresponden varias instrucciones máquina y no sólo una.

8.7 Conclusión

En este capítulo se ha descrito de forma clara y sencilla un computador en el nivel de complejidad de máquina convencional; es decir, desde el punto de vista de su repertorio de instrucciones en lenguaje máquina y de su lenguaje ensamblador. En primer lugar, se han analizado los modos de direccionamiento básico, que indican las formas en las que el procesador capta los operandos o almacena los resultados. Después se han dado unas ideas sobre los apartados o campos que suele contener una instrucción máquina (formatos). Para poder explicar adecuadamente los conceptos que subyacen bajo el lenguaje máquina y el lenguaje ensamblador, hemos descrito

un computador didáctico elemental (CODE-2), que se ha diseñado en función de su objetivo pedagógico y esperamos, al igual que lo fue su versión anterior ODE, sea una herramienta de gran utilidad para enseñar las ideas básicas que este capítulo presenta.

Debido a la existencia de toda una pléyade de lenguajes máquina y ensambladores (modelos de procesador distintos disponen de lenguajes distintos) hemos presentado uno de los intentos más alentadores para la unificación de la terminología de lenguajes ensambladores, como es el estándar IEEE 694.

En la última sección hemos incluido un análisis comparativo entre las características de los lenguajes máquina y ensambladores.

El siguiente capítulo (Capítulo 9) profundizará en muchos de los conceptos expuestos en el presente capítulo, pero desde el punto de vista del diseño del computador. Capítulos posteriores se centran en el análisis del computador en los niveles de máquina operativa (Capítulo 13) y máquina simbólica (Capítulo 14).

EJERCICIOS

Sugerencia: para confeccionar los programas de CODE seguir los siguientes pasos:

a) Describir las operaciones a realizar con un organigrama o con seudocódigo, adaptado a CODE.
b) Efectuar la asignación de memoria/registros para las constantes, variables y programa.
c) Redactar el programa en lenguaje máquina o ensamblador. (En caso de hacerlo en lenguaje máquina –hexadecimal–, previamente hacerlo con nemónicos).
d) Indicar claramente las instrucciones de uso del programa.

8.1 En una instrucción (de un determinado computador), que se encuentra en la posición de memoria *F7A2*, se incluye como operando el valor *37BC*. Suponiendo que en la posición *37BC* se encuentra la información *357D*, y que el registro *rD* (que actúa como registro índice) contiene *0034*. Indicar la dirección donde se encuentra el operando, suponiendo que se utiliza:

a) Direccionamiento inmediato.
b) Direccionamiento directo.
c) Direccionamiento indirecto relativo a memoria.
d) Direccionamiento indexado.
e) Direccionamiento relativo a contador de programa.

8.2 Hacer un programa para CODE que lea del puerto de entrada *IP1* un conjunto de N datos y contabilice y dé como resultado sucesivamente en el puerto de salida *OP2* el número de datos positivos, negativos y cero, existentes en el conjunto.

8.3 Dada una tabla de números que se inicia en la posición *5F3C* y acaba en la posición *F000,* hacer un programa para CODE-2 que proporcione por el puerto de salida *OP2* el máximo y el mínimo de esta tabla, y por el puerto de salida *OP1* las direcciones donde se encuentran dichos extremos.

8.4 Dada una tabla de números que se inicia en la posición *5F3C* y su último elemento tiene de contenido *5FFF* (que no aparece como dato en cualquier otra posición de la tabla), hacer un programa para CODE-2 que indique por el puerto *OP1* las direcciones relativas dentro de la tabla donde se encuentra el dato ubicado en la posición *A000* de la memoria.

8.5 Realizar una tabla comparando el número de registros utilizados y el tiempo consumido en la ejecución de los tres algoritmos descritos en la Sección 6.3.4 para comprobar si un número es par. De acuerdo con la tabla anterior ¿qué procedimiento es computacionalmente mejor?

8.6 Realizar un programa para CODE-2 que lea una serie de números del puerto de entrada *IP1,* y almacene los números pares en una zona de memoria que comience en la dirección *P*, y los números impares en otra zona que comience en la dirección *I*. El programa finaliza cuando se dé por la entrada el valor FFFF, teniendo entonces que facilitar por el puerto *OP2* los números, *NP* y *NI*, de datos pares e impares, respectivamente, leídos. Suponer que las direcciones de inicio de las tablas se encuentran en las posiciones A000 y A001 de memoria.

8.7 Dada una tabla de números enteros positivos que se inicia en la posición *B000* y finaliza en la *BFFF,* hacer un programa para CODE-2 que indique por el puerto *OP1* el número de datos de la tabla que son múltiplos de 4.

8.8 Hacer una rutina para CODE-2 que cambie la representación de los datos de una tabla. Los datos inicialmente se encuentran representados como números enteros en signo y magnitud, y hay que transformarlos a complemento a dos. Las posiciones inicial y final de la tabla se encuentran en las posiciones *A000* y *A001,* respectivamente, de memoria.

8.9 Hacer una rutina para CODE-2 que proporcione el cuadrado de un número entero comprendido entre 0 y 9. El dato inicial se encuentra en A001, y la rutina debe suministrar su cuadrado en la posición A002. *Sugerencia:* utilizar una tabla de consulta que contenga los cuadrados de los 10 números, y obtener los cuadrados consultando dicha tabla.

8.10 Entre la dirección *C000* y *CD88* de la memoria de CODE hay almacenada una lista de caracteres Unicode. Hacer un programa que busque la cadena "CODE" y muestre en el puerto de salida la dirección donde comienza.

8.11 De la dirección *I* = *B000* a la *F* = *BFFF* de la memoria de CODE-2 se encuentra una tabla de datos, que empiezan todos con un bit 0. (Es decir, el bit más significativo –MSB– es 0). Hacer un programa en código máquina que cambie los datos de la tabla añadiendo como bit MSB un bit de paridad (criterio par).

8.12 Realizar una estimación de la memoria y el tiempo que tardan en ejecutarse los programas descritos en la Sección 8.3.5. Suponer que la frecuencia de reloj de CODE-2 es de 1 GHz.

8.13 Realizar una rutina que produzca un retardo de 1 ms, suponiendo una implementación de CODE-2 cuya frecuencia de reloj sea 500 MHz.

8.14 En posiciones sucesivas de la memoria de CODE, a partir de la dirección DI = A000 se almacena una tabla de 500 datos. Realizar un programa que recorra la tabla, lleve, uno a uno, cada dato al puerto de salida OP1 y lo mantenga allí durante aproximadamente 3 segundos, antes de sobrescribirlo con el siguiente de la tabla. La versión de CODE-2 tiene una frecuencia de reloj de 1,6 GHz. *Sugerencia:* redactar una subrutina de retardo que realice la espera de 3 segundos.

8.15 Se tiene la siguiente instrucción *switch* de lenguaje C:

switch (k) {
case 0: g = j + n; break;
case 1: g = h + i; break;
case 2; g = h – i; break;
case 3; g = j – n; break;
};

Hacer una rutina en ensamblador de CODE-2 que implemente la instrucción anterior, suponiendo que los variables *g, i, j, k* y *n* tienen asignadas las posiciones de memoria *A000, A001, A002, A003* y *A004,* respectivamente.

8.16 A partir de la dirección 5A70 de la memoria de CODE-2 se tiene una lista encadenada. Cada elemento de la lista contiene 2 palabras: la primera es un dato, y la segunda constituye el puntero al dato siguiente (contienen la dirección del siguiente elemento de la lista). El último elemento de la lista tiene de puntero 0000. Los datos de la lista se encuentran dentro de ella ordenados de menor a mayor. En la dirección 3000 de memoria se encuentra un dato. Hacer un programa en ensamblador para incluir dicho dato en la lista, en el lugar que por su valor le corresponda. Suponer que el dato en 3000 siempre es mayor que el primer elemento de la lista y menor que el último elemento de la lista. *Sugerencia:* no es necesario cambiar de posición ningún dato de la lista, ni la posición del dato que está en 3000.

8.17 Suponiendo que los siete bits menos significativos del puerto de salida *OP3* de CODE-2 se encuentran conectados a un LED de 7 segmentos de ánodo común, tal como se indica en

la Figura 8.39. Hacer un programa en lenguaje ensamblador, que muestre por el indicador los valores hexadecimales correspondientes a los 4 bits menos significativos de los elementos de una tabla que comienza en la dirección *PT* = *B000*, y cuyo último valor (que no aparece como dato en cualquier otra posición de la tabla) es FFFF. *Sugerencia:* para activar adecuadamente los segmentos del LED utilizar una tabla de consulta (*lookup table*) ubicada en la memoria.

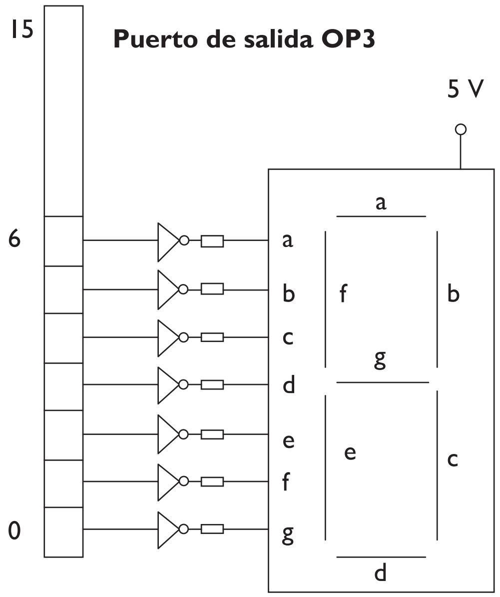


Figura 8.39. *Indicador LED conectado al puerto de salida OP3 de CODE-2.*

8.18 Se desea diseñar un sistema de control de temperatura con un procesador CODE-2. Los 6 bits más significativos del puerto de entrada IP3 están conectados a la salida de un conversor A/D que proporciona la temperatura actual del recinto. Los 6 bits menos significativos de dicho puerto se utilizan para introducir el valor de la temperatura deseada ("consigna" de temperatura). El sistema debe disponer de 4 salidas, conectadas todas ellas al puerto de salida OP3, con las siguientes conexiones y funciones (entre paréntesis se especifica el bit correspondiente):

O1(11): si se proporciona en este bit 1 se acciona una electroválvula que abre la conducción de aire; por el contrario con 0 se cierra.

O1(10): si se saca en el 1 se cierra un relé que conecta el calefactor (calor); y debe estar a 0 para que se abra el relé; es decir, se desconecte el calefactor.

O1(9): actúa igual que el bit de salida O1(10), pero para conectar o desconectar un refrigerador.

O(6:0): estos 7 bits están conectados directamente a un indicador de 7 segmentos como el de la Figura 8.39 (cuando la entrada a un elemento del indicador es 1, éste se activa). Se desea que en el indicador aparezcan las siguientes letras:

A, si el sistema de caldeo/refrigeración está apagado.

F, si el sistema está enfriando.

C, si el sistema está calentando.

Realizar un programa en ensamblador que realice el control de temperatura, teniendo en cuenta lo siguiente:

a) El sistema debe actuar calentando o enfriando cuando la temperatura de consigna se diferencie (positiva o negativamente, respectivamente) en 1 ó más grados con respecto a la temperatura ambiente.
b) La conexión de aire debe actuar si y sólo si hay refrigeración o caldeo.
c) El indicador de siete segmentos debe marcar la acción que se esté realizando.

BIBLIOGRAFÍA

[Cap01] Carpinelli, J. D.: *Computer Systems. Organization & Architecture,* Addison Wesley, 2001.

[Dow00] Dowsing, R. D.; Woodhams, F. W. D.; Marshall, I.: *Computers from logic to architecture;* 2nd Edt. MacGraw-Hill, 2000.

[Fer98] Fernández, G.: *Conceptos básicos de arquitectura y sistemas operativos;* Sistemas y Servicios de Comunicación, 1998.

[Ham03] Hamacher, C.; Vranesic, Z.; Zaky, S.: *Organización de computadores*, 5ª Ed., MacGraw-Hill, 2003.

[Hay98] Hayes, J. P.: *Computer Architecture and Organization,* 3ª Ed. McGraw-Hill, 1998.

[IEE85] IEEE: *IEEE Standard for Microprocessor Assembly Language*. IEEE Std. 694-1985. The Institute of Electrical and Electronics Engineers, Inc. New York., June 30, 1985.

[Kno90] Knott, G.; Waites, N.; Callaghan, P.; Ellison, J.: *Computer Studies for BTEC*. 2nd. Edt. Busines Education Pub. Ltd., 1990.

[Lop05] López, J. I.; Pelayo, F. J.; Prieto, A.; Ortega,J.: *Segmentación de Cauce del Computador Didáctico Elemental CODE-2*; Actas de las XVI Jornadas de Paralelismo, pp. 725-732, Thomson, I Congreso Español de Informática (CEDI2005) Granada, 13-16 Septiembre 2005.

[Man97] Mano, M.; Kime, C.: *Logic and computer design fundamentals.* Prentice Hall, 1997.

[Mur00] Murdocca, M. J.; Heuring, V. P.: *Principles of Computer Architecture,* Prentice Hall, 2000.

[Pat00] Patterson, D. A., Hennessy, J. L.: *Estructura y diseño de computadores;* Editorial Reverté, 2000.

[Pat01] Patt, Y. N.: *Introduction to computing systems,* McGraw-Hill, 2001.

[Pri02a] Prieto, A.; Pelayo, F. J.; Gómez-Mula, F.; Ortega, J.; Cañas, A.; Martínez, A; Fernández, F. J.: *Un computador didáctico elemental (CODE-2),* Actas de las VIII Jornadas de Enseñanza Universitaria de la Informática (JENUI'2002), pp. 117-124, Cáceres, 10-12 Julio 2002.

[Pri02b] Prieto, A.; Pelayo, F. J; del Pino, B.; Ortega, J.; Pomares, H.; Gómez-Mula, F.; Cañas,A.; Díaz, A.: *CODE-2: Un Computador Didáctico Elemental,* Actas de las XIII Jornadas de Paralelismo, pp. 73-77, Lérida, 9-11 Septiembre 2002.

[Pri05] Prieto, A.; Prieto, B.: *Conceptos de Informática,* Serie Shaum, McGraw-Hill, 2005.

[Sim97] Sima, D.; Fountain, T.; Kacsuk, P.: *Advanced Computer Architectures: A design space approach,* Addison-Wesley, 1997.

[Sta06] Stallings, W.: *Organización y arquitectura de computadores*. 7ª Edc. Prentice Hall, 2006.

[Tan00] Tanenbaum, A. S.: *Structured Computer Organization,* 5ª Edc. Prentice Hall, 2006.

[Wil94] Wilkinson, B.: *Computer architecture: design and performances,* Prentice Hall, 1994.

[Zar96] Zarghan, M. R.: *Computer Architecture,* Prentice Hall, 1996.

CAPÍTULO 9

El computador a nivel de micromáquina

En este capítulo, dedicado a realizar una introducción a la estructura de computadores, se describe el funcionamiento y diseño de las unidades centrales de un computador: unidad de tratamiento (camino de datos), unidad de control, memoria; así como la organización de las entradas/salidas. Según se analizan las distintas unidades se realiza el diseño del computador CODE-2; para lo cual se utilizarán las técnicas de diseño lógico presentadas en los Capítulos 5 y 6. La descripción del presente capítulo corresponde al **nivel de micromáquina,** también denominado **nivel de transferencia entre registros** o **RTL** (*Register Transfer Level*).

El diseño de cada unidad del computador se va a realizar de forma descriptiva e intuitiva a partir de una descripción detallada explícita (caso por caso). No obstante, existen métodos sistemáticos para el diseño de procesadores. Así, por ejemplo, a partir de una descripción (denominada **diagrama de máquina de estados algorítmica**, **ASM**, *Algorithm-State-Machine*) del comportamiento del procesador a nivel de micromáquina, se puede obtener de forma sistemática su implementación de acuerdo con el **modelo FSMD** (**máquina de estados finitos con camino de datos**, *FSM with a Datapaph*). De esta forma se puede diseñar un procesador a partir de su repertorio de instrucciones. Es más, existen herramientas con las que a partir de la descripción de la estructura o del comportamiento del sistema en un **lenguaje de descripción del hardware** (**HDL***, Hardware Description Language*), tal como **VHDL** (*Very High speed integrated circuits Hardware Desciption Language*) o **Verilog**, se puede obtener automáticamente su diseño físico, sin apenas la intervención del diseñador. En la referencia [Dia02] puede encontrarse el diseño de CODE-2 utilizando estas técnicas.

9.1 La aritmética en el computador

El conjunto de elementos de un procesador no dedicados al control se suele denominar, dependiendo de los autores, **unidad-aritmético lógica** o **unidad de procesamiento** o **unidad de tratamiento** o **camino de datos**. Usualmente el término unidad aritmético-lógica o ALU se reserva para los circuitos específicos que realizan este tipo de operaciones. En este capítulo preferimos utilizar el término **unidad de tratamiento**, e incluye la unidad aritmética-lógica propiamente dicha (ALU), acumulador, registros auxiliares (para resultados intermedios o direcciones), multiplexores, etc.

En la ALU se suelen realizar operaciones muy sencillas como rotaciones y desplazamientos de uno o varios bits a la vez del acumulador (y posiblemente de algún otro registro interno),

comparaciones, sumas, restas, multiplicaciones y divisiones. En el Capítulo 5 se vieron algunos circuitos básicos para hacer sumas (Sección 5.4.2), comparaciones (Sección 5.4.3), etc. Para completar esta visión, en la Sección 9.1.1 veremos algoritmos para realizar las operaciones básicas con números enteros representados en la forma que se indicó en la Sección 4.5.1. En la Sección 9.1.2 incluimos algunas ideas básicas sobre cómo se diseñan los elementos de la unidad aritmética que implementan los algoritmos citados. La Sección 9.1.3 describe los algoritmos básicos para operar con datos reales; es decir, representados en coma flotante, tal y como se estudió en la Sección 4.5.2. Como complemento del apartado anterior, en la Sección 9.1.4 se hace un apunte sobre la implementación de la unidad aritmética de coma flotante. En la Sección 9.2, para mejor comprender cómo funciona a nivel de micromáquina el camino de datos de un computador, describiremos el diseño del CODE-2.

9.1.1 Operaciones básicas con datos enteros

Considérense dos números:

N1, con representación interna *s1 m1*

N2, con representación interna *s2 m2*

donde *s1* y *s2* representan los signos, mientras que *m1* y *m2* los valores absolutos de dichos números.

Al resultado de la operación entre *N1* y *N2* le denominaremos:

Nr, y quedará almacenado como *sr mr*

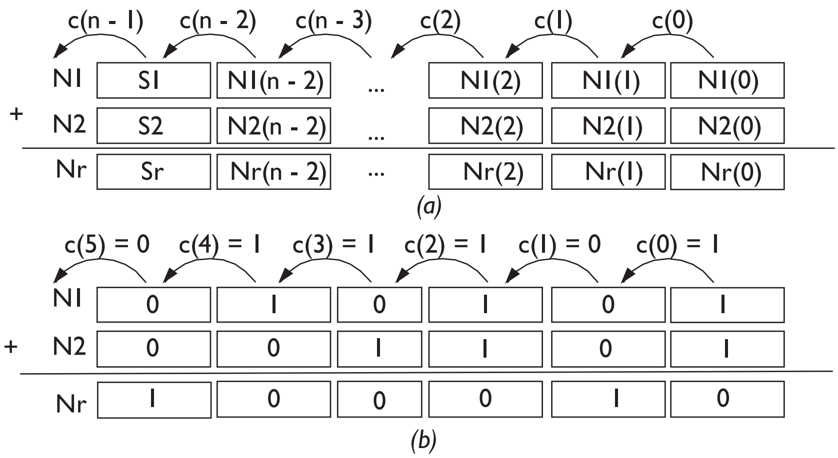


Figura 9.1. *(a) Notación seguida para denominar los acarreos y las distintas cifras que intervienen en una operación; (b) ejemplo de suma de dos números 010101 + 001101) de n = 6 bits (incluyendo bit de sino).*

Suponemos que el número total de bits de cada dato es *n:* el primero de signo y el resto (es decir, n – 1 bits) para su valor. Al operar con dos datos se van produciendo, además de los bits de resultados, acarreos (que pueden ser 0 ó 1), tal como se muestra en la Figura 9.1. Denominamos c_0 al acarreo producido por los bits menos significativos de *m1* y *m2* (bits de orden *0*), c_1 al acarreo producido por los bits de orden *1*, etc. El acarreo producido por los bits más significativos del valor absoluto (bits de orden *n – 2*) será c_{n-2}. Los algoritmos (y los circuitos que los

implementan) siempre producen un resultado, aunque se haya producido un desbordamiento; y en este caso dicho resultado es incorrecto. Ello quiere decir que siempre el algoritmo (o circuito aritmético) debe generar un bit de desbordamiento, v, y hacerlo igual a 1 en el caso de que se produzca éste. En otras palabras, la ALU siempre generará un resultado, pero éste únicamente será correcto si $v = 0$.

Vamos a considerar las operaciones más elementales en las distintas representaciones para datos enteros vistas en la Sección 4.6.1.

9.1.1.1 Representación en signo y magnitud

A continuación se analiza la forma de operar en el formato de signo y magnitud. El algoritmo de la suma es el siguiente:

Suma $N_r \leftarrow N_1 + N_2$

si $s1 = s2$; hacer $mr \leftarrow m1 + m2$; $sr \leftarrow s1$

si $s1 \neq s2$

si $m1 \geq m2$ hacer $mr \leftarrow m1 - m2$; $s_r \leftarrow s_1$

si $m1 < m2$ hacer $mr \leftarrow m2 - m1$; $s_r \leftarrow s_2$

Se produce desbordamiento (el valor absoluto del resultado no cabe en n – 1 bits) si $c_{n-2} = 1$; es decir, el bit de desbordamiento resulta ser: $v \leftarrow c_{n-2}$.

La suma (y resta) propiamente dicha de (los módulos de los) números binarios ($mr \leftarrow m1 \pm m2$) se efectúa tal y como se indicó en el Apéndice A (Sección A1.2).

El algoritmo de la resta es el siguiente:

Resta $N_r \leftarrow N_1 - N_2$

si $s1 = s2$

si $m1 \geq m2$ hacer $mr \leftarrow m1 - m2$; $s_r \leftarrow s_1$

si $m1 < m2$ hacer $mr \leftarrow m2 - m1$; $s_r \leftarrow \overline{s_1}$

si $s1 \neq s2$; hacer $mr \leftarrow m1 + m2$; $s_r \leftarrow s_1$

Se produce desbordamiento (el valor absoluto del resultado no cabe en n – 1 bits) si $c_{n-2} = 1$; es decir, hacer $v \leftarrow c_{n-2}$.

Recuérdese que $\overline{s_1}$ o s_1' denota el complemento de s_1 (1, si s_1 es 0, ó 0, si s_1 es 1). Puede observarse que la resta presenta la misma complejidad que la suma.

Obsérvese que los algoritmos anteriores implican la realización de tres operaciones sucesivas:

1) Establecer si los signos son iguales o no.
2) En caso de que los signos sean distintos, identificar el número con mayor valor absoluto y el de menor valor absoluto.
3) Realizar el cálculo del valor absoluto del resultado y establecer el signo del resultado.

Cada una de las operaciones anteriores depende de la anterior, por lo que su implementación física requerirá el consumo de varios ciclos de reloj o la utilización de circuitos muy complejos. En definitiva, las operaciones más básicas que realiza la ALU (suma o resta de dos números) no es nada eficiente.

El algoritmo de la multiplicación es el siguiente:

Multiplicación	$\mathbf{N_r \leftarrow N_1 \cdot N_2}$
Hacer	$mr \leftarrow m1 \cdot m2;\ v \leftarrow c_{n-2}$
	$s_r \leftarrow s_1 \oplus s_2$

El producto de los módulos de los números binarios ($mr \leftarrow m1 \cdot m2$) se efectúa tal y como se indica en el Apéndice A (Sección A1.2). Recuérdese que el símbolo ⊕ representa la función lógica ***exclusive OR***. Esta función produce un 1 lógico como resultado, si las entradas son distintas; y un 0 lógico en caso contrario (Sección 5.3). Por lo general, para almacenar el resultado de la multiplicación se utiliza un número de bits *n* mayor que el número de bits de los operandos, con objeto de eliminar las posibilidades de que se produzca desbordamiento.

El algoritmo de la división de enteros representados en signo y magnitud es:

División	$\mathbf{N_r \leftarrow N_1/N_2}$
Hacer	$mr \leftarrow m1/m2;\ v \leftarrow c_{n-2}$
	$s_r \leftarrow s_1 \oplus s_2$

El cociente de números binarios se efectúa tal y como se indica en el Apéndice A (Sección A1.2).

EJEMPLO 9.1

Supongamos que $n = 8$, y que $N_1 = 10111001$ y $N_2 = 0111\ 0101$ son dos números representados en signo y magnitud. Obtener $N_1 + N_2$ y $N_1 - N_2$.

Se verifica que:

- *$s1 = 1$ (es decir, N_1 es negativo), $m1 = 011\ 1001$*
- *$s2 = 0$ (es decir, N_2 es positivo), $m2 = 111\ 0101$*
- *$m2 > m1$*

$N_1 + N_2$

Como los signos son distintos, realmente tendremos que restar, el mayor del menor; es decir:

$N_r = (|N_2| - |N_1|)$

$s_r = s_2 = 0$

$m_r = m2 - m$, es decir:

$m_2 \rightarrow$		1	1	1	0	1	0	1
$- m_1 \rightarrow$		0	1	1	1	0	0	1
adeudos $\rightarrow$	0	1	1	1				
$m_r \rightarrow$		0	1	1	1	1	0	0

EJEMPLO 9.1 (Cont.)

Como $v \leftarrow c_6$ *(el último adeudo) es 0, no hay desbordamiento, siendo el resultado anterior correcto.*
Luego el resultado es: $N_r = 0011\ 1100$

$N_1 - N_2$

Como los signos son distintos, realmente tendremos que sumar, el mayor del menor; es decir:

$N_r = N_1 - N_2 = -(|N_1| + |N_2|)$

$s_r = s_1 = 1$

$m_r = m1 + m2$

acarreos →	1	1	1			1	
m_1 →		0	1	1	1	0	0	1
+ m_2 →		1	1	1	0	1	0	1
m_r →		0	1	0	1	1	1	0

Como $v \leftarrow c_6$ *(el último acarreo) es 1, hay desbordamiento, siendo el resultado anterior incorrecto. Luego el resultado es desbordamiento.*

9.1.1.2 Representación en complemento a uno

Las operaciones básicas con datos representados en la notación de complemento a uno se realizan como sigue.

Algoritmo de la suma:

Suma $N_r \leftarrow N_1 + N_2$

Esta operación se realiza sumando todos los bits del dato, *incluyendo los signos*. El acarreo de mayor orden (el generado por los signos), debe sumarse al resultado. Es decir:

Hacer $[sr\ mr] \leftarrow [s1\ m1] + [s2\ m2] + c_{n-1}$

Se produce desbordamiento si $c_{n-1} \neq c_{n-2}$; esto es, hacer $v \leftarrow c_{n-1} \oplus c_{n-2}$.

La suma de números binarios se efectúa tal y como se indica en el Apéndice A (Sección A1.2). Obsérvese que, a diferencia de la representación en signo y magnitud, ahora no hay que tomar ninguna decisión previa antes de realizar el cálculo propiamente dicho: la suma (independientemente de los signos de los operandos) no requiere tres fases, por lo que será fácil de implementar, y resulta rápida ya que se puede realizar en un ciclo de reloj. El algoritmo de la resta es:

Resta $Nr \leftarrow N1 - N2$

Hacer $[sr\ mr] \leftarrow [s1\ m1] - [s2\ m2] - c_{n-1}$

Se produce desbordamiento si $c_{n-1} \neq c_{n-2}$; es decir, hacer $s_r \leftarrow c_{n-1} \oplus c_{n-2}$.

La resta de números binarios se efectúa tal y como se indica en el Apéndice A (Sección A1.2), pudiéndose realizar sumando al minuendo el complemento a uno del sustraendo (Sección A1.3). También la implementación de la resta resulta fácil y rápida.

El procedimiento para multiplicar números representados en complemento a uno es el siguiente:

> ***Multiplicación*** $N_r \leftarrow N_1 \cdot N_2$
> Si $s_1 = 1$, hacer complemento a uno de m_1
> Si $s_2 = 1$, hacer complemento a uno de m_2
> Hacer $m_r \leftarrow m_1 \cdot m_2$; $s_r \leftarrow s_1 \oplus s_2$; $v \leftarrow c_{n-2}$
> Si $s_r = 1$, hacer complemento a uno de m_r

El producto de números binarios ($m_r \leftarrow m_1 \cdot m_2$) se efectúa tal y como se indica en el Apéndice A (Sección A1.2).

El algoritmo de la división es:

> ***División*** $N_r \leftarrow N_1/N_2$
> Si $s_1 = 1$, hacer complemento a uno de m_1
> Si $s_2 = 1$, hacer complemento a uno de m_2
> Hacer $m_r \leftarrow m_1/m_2$; $s_r \leftarrow s_1 \oplus s_2$; $v \leftarrow c_{n-2}$
> Si $s_r = 1$, hacer complemento a 1 de m_r

El cociente de números binarios ($m_r \leftarrow m_1/m_2$) se efectúa tal y como se indica en el Apéndice A (Sección A1.2).

EJEMPLO 9.2

Supongamos que $n = 8$, y que tenemos dos números $N_1 = 11000110$ y $N_2 = 0111\ 0101$ que están representados en complemento a uno. Obtener $N_1 + N_2$ y $N_1 - N_2$.

$N_1 + N_2$

Sencillamente sumamos todos los bits, incluyendo el de signo, y el acarreo final (c_7) se lo sumamos al resultado anterior:

acarreos →	1	1				1			
N_1 →		1	1	0	0	0	1	1	0
+ N_2 →		0	1	1	1	0	1	0	1
		0	0	1	1	1	0	1	1
+ c_7 →									1
N_r →		0	0	1	1	1	1	0	0

Como $c_7 = c_6$ el resultado obtenido es correcto (no hay desbordamiento); con lo que $N_r = 00111100$.

EJEMPLO 9.2 (Cont.)

$N_1 - N_2$

Sencillamente, sumamos al minuendo el complemento a uno del sustraendo. La suma la realizamos según el algoritmo anterior:

acarreos (c_1) →	1	0			1	1	1		
N_1 →		1	1	0	0	0	1	1	0
+ $C_1(N_2)$ →		1	0	0	0	1	0	1	0
		0	1	0	1	0	0	0	0
+ c_7 →									
N_r →									

Como $c_7 \neq c_6$ el resultado obtenido es incorrecto: hay desbordamiento.

9.1.1.3 Representación en complemento a dos

Como a continuación se indica, en la notación de complemento a dos, las operaciones básicas se simplifican notablemente.

El algoritmo de la suma es el siguiente:

Suma $N_r \leftarrow N_1 + N_2$

Esta operación se realiza, independientemente de los signos y valores absolutos de los datos, sumando todos los bits del dato, *incluyendo los signos*. El acarreo de mayor orden (el generado por los signos), a diferencia de la representación en complemento a uno, no se considera. Es decir:

Hacer *[sr mr]* ← *[s1 m1]* + *[s2 m2]*

Se produce desbordamiento si $c_{n-1} \neq c_{n-2}$; es decir, hacer $v \leftarrow c_{n-1} \oplus c_{n-2}$

La suma de números binarios se efectúa tal y como se indica en el Apéndice A (Sección A1.2). Observamos que la representación en complemento a dos es la más útil para implementar la suma (véase Sección A1.3), aunque el sistema para hacer la conversión de signo y magnitud a complemento a dos (y viceversa) resulta más complejo que en el caso de la representación en complemento a uno. El algoritmo para la resta es el siguiente:

Resta $N_r \leftarrow N_1 - N_2$

Hacer *[sr mr]* ← *[s1 m1]* – *[s2 m2]*

Se produce desbordamiento si $c_{n-1} \neq c_{n-2}$; es decir, hacer $v \leftarrow c_{n-1} \oplus c_{n-2}$

La resta de números binarios se efectúa tal y como se indica en el Apéndice A (Sección A1.2); pudiéndose realizar sumando al minuendo el complemento a uno del sustraendo. Observamos que la representación en complemento a dos es la más útil para implementar la resta (véase Sección A1.3), aunque el sistema para hacer la conversión de signo y magnitud a complemento a dos (y viceversa) resulta más complejo que en el caso de la representación en complemento a uno. La multiplicación se realiza de acuerdo con el siguiente procedimiento:

> ***Multiplicación*** $\boldsymbol{N_r \leftarrow N_1 \cdot N_2}$
> Si $s_1 = 1$, hacer complemento a dos de m_1
> Si $s_2 = 1$, hacer complemento a dos de m_2
> Hacer $m_r \leftarrow m_1 \cdot m_2$; $s_r \leftarrow s_1 \oplus s_2$; $v \leftarrow c_{n-2}$
> Si $s_r = 1$, hacer complemento a dos de m_r

El producto de números binarios ($m_r \leftarrow m_1 \cdot m_2$) se efectúa tal y como se indica en el Apéndice A (Sección A1.2). El algoritmo de la división es:

> ***División*** $\boldsymbol{N_r \leftarrow N_1/N_2}$
> Si $s_1 = 1$, hacer complemento a 2 de m_1
> Si $s_2 = 1$, hacer complemento a 2 de m_2
> Hacer $m_r \leftarrow m_1/m_2$; $s_r \leftarrow s_1 \oplus s_2$; $v \leftarrow c_{n-2}$
> Si $s_r = 1$, hacer complemento a 2 de m_r

El cociente de números binarios ($m_r \leftarrow m_1/m_2$) se efectúa tal y como se indica en el Apéndice A (Sección A1.2).

EJEMPLO 9.3

Supongamos que $n = 8$, y que tenemos dos números $N_1 = 11000111$ y $N_2 = 0111\ 0101$ que están representados en complemento a dos. Obtener $N_1 + N_2$ y $N_2 - N_1$.

$N_1 + N_2$

Sencillamente sumamos todos los bits, incluyendo el de signo, y el acarreo final (c_7) lo despreciamos:

acarreos (c_i) →	1	1				1	1	1	
N_1 →		1	1	0	0	0	1	1	1
+ N_2 →		0	1	1	1	0	1	0	1
N_r →		0	0	1	1	1	1	0	0

Como $c_7 = c_6$ el resultado obtenido es correcto (no hay desbordamiento); con lo que $N_r = 00111100$.

EJEMPLO 9.3 (Cont.)

$N_2 - N_1$

Sencillamente sumamos al minuendo el complemento a dos del sustraendo. La suma la realizamos según el algoritmo anterior:

acarreos (c_i) →	0	0	1	1	1			
N_2 →		1	0	0	0	1	0	1	0
+ C_2 (N_1) →		0	0	1	1	1	0	0	1
N_r →		1	1	0	0	0	0	1	1

Como $c_7 = c_6 = 0$ el resultado obtenido es correcto (no hay desbordamiento), y N_r= 1100 0011.

9.1.1.4 Representación en BCD

Para operar con números BCD se han de utilizar circuitos específicamente diseñados para este código. Por ejemplo, para sumar en BCD números con *n* dígitos decimales, en paralelo, se utilizaría un sumador con *n* módulos, uno para cada dígito decimal, teniendo cada módulo 9 entradas (4 para cada uno de los dos dígitos decimales y 1 para el acarreo previo) y 5 salidas (4 correspondientes al dígito decimal de suma y 1 para el acarreo de salida). No obstante, para sumar números BCD es corriente utilizar sumadores binarios; para ello hay que realizar correcciones adecuadas en las salidas generadas por los sumadores. Veamos qué ajustes deben realizarse. Consideremos algunos ejemplos al respecto. Sean los dígitos decimales siguientes:

$$1)_{10} = 0001)_{BCD}; \qquad 5)_{10} = 0101)_{BCD};$$

$$7)_{10} = 0111)_{BCD}; \qquad 9)_{10} = 1001)_{BCD}$$

y realicemos las siguientes sumas utilizando sumadores binarios:

1	0001	5	0101	7	0111	9	1001
+ 5	0101	+ 7	0111	+ 9	1001	+ 9	1001
6	0110	12	1100	16	**1**0000	18	**1**0010

Cuando se suman en binario los dígitos BCD 1 y 5 el resultado es el dígito BCD 6, que es correcto. Luego en este caso no hay que realizar corrección alguna. Generalizando, cuando la suma es menor que diez, el resultado que se obtiene es correcto.

Al sumar los dígitos BCD 5 y 7 se obtiene un resultado incorrecto por un doble motivo: debería producirse un acarreo y el dígito suma debería ser 2. En efecto, el resultado que se obtiene no es un carácter BCD. Es fácil comprobar que si se suma 6 al resultado obtenido, se obtiene ya el valor correcto. En efecto,

$$\begin{array}{r} 1100 \\ +\ \underline{0110} \\ 1\ 0010 \end{array}$$

Al sumar los dígitos BCD 7 y 9 (o 9 y 9) el resultado es el dígito BCD 0 (o 2), que es incorrecto, y se genera un acarreo, que es correcto. De nuevo es inmediato que sumando 6 al resultado obtenido, se genera el resultado correcto. Es decir, cuando la suma es mayor o igual a diez, el valor correcto se obtiene sumando 6 a la suma binaria obtenida.

Otras operaciones BCD realizadas con circuitos para aritmética binaria también requieren correcciones ulteriores, similares a las de la suma. En la resta, si el resultado no es una cifra BCD, se le resta 6.

EJEMPLO 9.4

Supongamos un computador con palabras de 4 bits (n = 4). Efectuar las siguientes operaciones: 1 + 5, 7 + 2; (–7) + 3; 5 + (–6) y 6 + (–3); en decimal, en signo y magnitud en complemento a uno y en complemento a 2.

Los números en binario natural son: 1 = 001; 5 = 101; 7 = 111; 2 = 010; –7 = –111; 3 = 011; 6 = 110; –3 = -011.

Decimal:

1 + 5 = 6; 7 + 2 = 9 (con 3 bits de precisión, en binario habría desbordamiento)

(–7) + 3 = –4; 5 + (–6) = –1; 6 + (–3) = 3

Signo y magnitud

Recordar que el primer bit representa al signo del número

	0001	0111	1111	0110	0110
	+ 0101	+ 0010	– 0011	– 0101	– 0011
Acarreos o adeudos →	0001-	0110-	0000-	0001-	0111-
*Resultado (**Nr**)* →	0110	*Desb.*	1100	1001	0011

Complemento a uno

	0001	0111	1000	0101	0110
	+ 0101	+ 0010	– 0011	– 1001	– 1100
Acarreos o adeudos →	0001-	0110-	0000-	0001-	1100-
*Resultado (**Nr**)* →	0110	*Desb.*	1011	1110	(1)0010
					$c_{n-1} \rightarrow$ 1
					0011

Complemento a dos

	0001	0111	1001	0101	0110
	+ 0101	+ 0010	+ 0011	– 1010	1101
Acarreos o adeudos →	0001-	0110-	0000-	0000-	1100-
*Resultado (**Nr**)* →	0110	*Desb.*	1100	1111	0011

EJEMPLO 9.5

Efectuar en BCD empaquetado (8 bits) las siguientes sumas de números decimales:

$$17 + 32;\ 29 + 63;\ 38 + 87$$

Los datos en código BCD son los siguientes:

$17)_{10} = 0001\ 0111)_{BCD}$; $32)_{10} = 0011\ 0010)_{BCD}$; $29)_{10} = 0010\ 1001)_{BCD}$;
$63)_{10} = 0110\ 0011)_{BCD}$; $38)_{10} = 0011\ 1000)_{BCD}$; $87)_{10} = 1000\ 0111)_{BCD}$

Para hacer la suma de los datos BCD primero realizamos una suma binaria y después ajustamos el resultado para que quede en BCD; es decir, si el resultado no es una cifra BCD (no es menor de 1010), sumamos $6)_{10} = 0110)_2$.

	0001 0111	0010 1001	0011 1000
	+ 0011 0010	+ 0110 0011	+ 1000 0111
Suma binaria →	0100 1001	1000 1100	1011 1111
Ajuste BCD →		+ 0110	+ 0110 0110
Resultado BCD →	**0100 1001**	**1001 0010**	**0001 0010 0101**
Resultado decimal →	49	92	125

Con lo visto en esta sección queda claramente puesto de manifiesto el motivo de por qué preferentemente se suele utilizar en computadores las notaciones en complementos: las operaciones aritméticas resultan muy fáciles de implementar y rápidas; como se pone de manifiesto en la siguiente sección.

9.1.2 Implementación de la aritmética de enteros

En esta sección sólo se pretende dar unas ideas muy básicas sobre la implementación de los algoritmos para operar con datos de tipo entero. El lector interesado puede acudir a las referencias [Kor93], [Omo94], [Pat00], [Sta00],[Swa90] y [Wal90] donde se pueden encontrar numerosos detalles sobre este tema.

9.1.2.1 Suma y resta

Cuando realizamos manualmente una suma de dos números, la realizamos por *etapas,* empezando por el dígito de la derecha o menos significativo (etapa 0) y acabando con el más significativo. En la Figura 9.2 se muestra un ejemplo de la suma *3732 + 1584*.

Según vimos en la Sección 4.5.2, la suma binaria de dos números *sin signo* se puede implementar en el nivel de lógica digital de la misma forma: un sumador de $n - 1$ bits se implementaría con $n - 1$ etapas. El circuito completo, aparte de las $2 \cdot (n - 1)$ entradas y $n - 1$ salidas de los datos, tiene una entrada de acarreo, c_{-1}, y una salida del acarreo del bit más significativo, c_{m-1}. Ahora bien, el circuito anterior es válido para realizar la suma binaria sin más, ya que no genera los bits de signo y de desbordamiento. En el caso de utilizar datos representados

en signo y magnitud, habría que realizar un sistema más complejo que implementase las tres fases indicadas en la Sección 9.1.1 sumador binario visto sólo serviría para la suma de los valores absolutos en el caso de que los signos fuesen iguales.

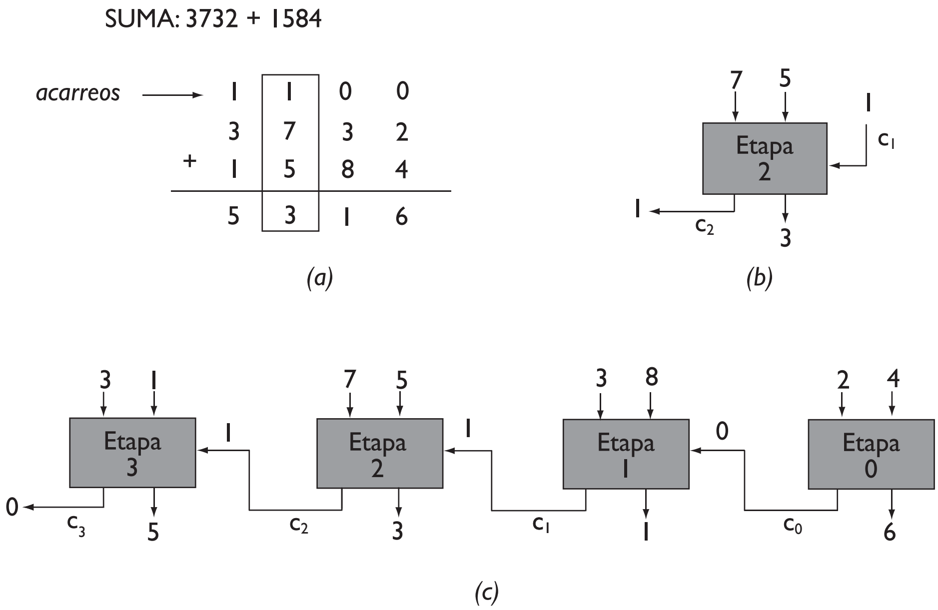


Figura 9.2. *Ejemplo de realización de una suma decimal, y cómo se realiza ésta por etapas: (a) suma; (b) etapa 2; (c) sumador completo de 4 dígitos decimales.*

La **suma y resta en complemento a uno** se puede realizar con la estructura indicada en la Figura 9.3. En este esquema el bloque sumador, representa al sumador binario de la Sección 5.5, N1 y N2 representan a los bits de cada dato (incluyendo el de signo), y la función exclusive OR se realiza entre el valor de la señal R/S y cada uno de los bits de N2. La señal R/S es una señal de control que cuando es 0 se hace la suma N1 + N2, y cuando es 1, se realiza la resta; como vemos a continuación. Recuérdese que, según vimos en la sección anterior, la operación a realizar es:

Hacer [sr mr] $\leftarrow$ *[s1 m1] + [s2 m2] +* c_{n-1}
Se produce desbordamiento si $c_{n-1} \neq c_{n-2}$; *es decir, hacer* $v \leftarrow c_{n-1} \oplus c_{n-2}$

Esto es exactamente lo que hace el circuito, ya que el acarreo c_{n-1} está conectado al c_{-1} con lo que se sumará a los dos datos. Por otra parte se genera un bit de desbordamiento en la forma requerida ya que v se activa con el resultado de la función exclusive-OR entre c_{n-1} y c_{n-2}.

La resta se realiza sumando al minuendo (N1) el complemento a uno del sustraendo. En efecto, cuando R/S = 1, se efectúa la operación exclusive-OR de los bits de N2 con el valor 1; esto equivale a complementar cada uno de los bits de N2, con lo que la suma se efectuará entre N1 y el complemento a uno de N2, que es lo que se pretendía.

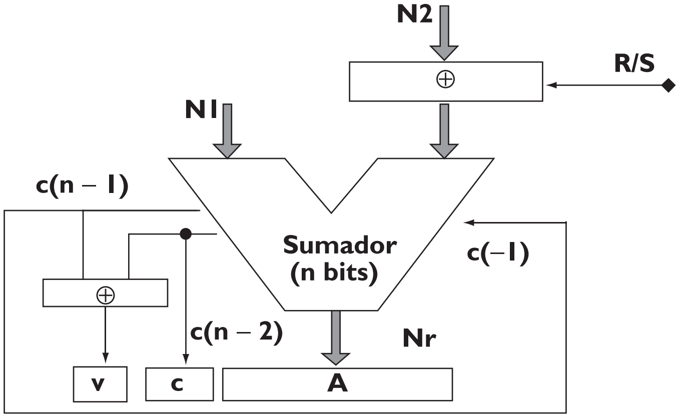


Figura 9.3. *Arquitectura para realizar la suma y la resta de datos representados en complemento a uno.*

La **suma y resta en complemento a dos** se puede realizar utilizando el esquema de la Figura 9.4; que es muy similar al del caso de complemento a uno. Recuérdese que, según vimos en la sección anterior, la operación a realizar es:

Hacer [sr mr] ← [s1 m1] + [s2 m2] *Se produce desbordamiento si* $c_{n-1} \neq c_{n-2}$*; es decir, hacer* $v \leftarrow c_{n-1} \oplus c_{n-2}$

En este caso no hay que sumar al resultado el acarreo c_{n-1}. El desbordamiento se genera también como antes: v se activa con el resultado de la función exclusive-OR entre c_{n-1} y c_{n-2}.

La resta se realiza sumando al minuendo (N1) el complemento a dos del sustraendo. En efecto, cuando R/S = 1, se efectúa la operación exclusive-OR de los bits de N2 con el valor 1; esto equivale a complementar cada uno de los bits de N2. Para hacer el complemento a dos, además debemos sumar 1 al complemento a uno de N2. El circuito exactamente no hace eso, en realidad en la resta se suma 1 al resultado a través de la línea c_{-1} ya que en ese caso R/S = 1. El resultado es completamente válido ya que da lo mismo sumar 1 al segundo sumando que al resultado.

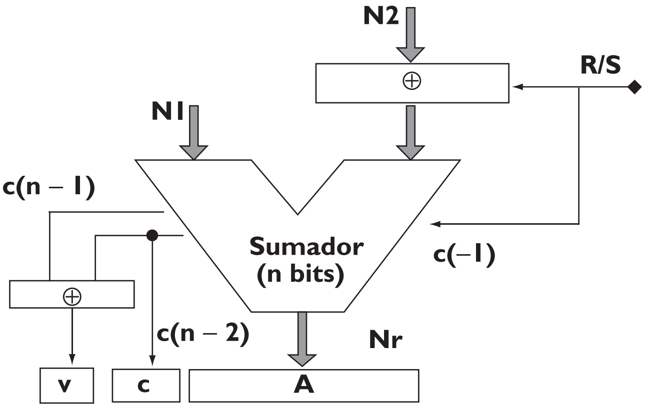


Figura 9.4. *Arquitectura para realizar la suma y la resta de datos representados en complemento a dos.*

Obsérvese que el circuito para sumar o restar en complemento a dos es muy sencillo, y la operación se puede hacer en un ciclo de reloj.

9.1.2.2 Multiplicación secuencial para enteros sin signo

Hasta la década de los 80 la mayoría de procesadores no disponían de circuitos específicos para implementar las operaciones de multiplicación y división de números enteros, no existiendo en ellos instrucciones máquina específicas para realizarlas. En estos computadores las operaciones citadas se implementaban por software; concretamente se incluían rutinas, que solían formar parte de los macroensambladores (véase Sección 8.6) para realizar el producto y la división utilizando instrucciones del repertorio del lenguaje máquina. Esta solución es totalmente válida, pero los programas que necesiten utilizar estas operaciones se ejecutarán con lentitud.

Hay numerosos algoritmos para realizar por hardware la multiplicación y la división. Los más rápidos son más complejos de describir, y en esta sección únicamente vamos a analizar dos algoritmos muy sencillos para ilustrar al lector sobre el tema. En la referencia [Raf90] se describen a nivel introductorio otros algoritmos más rápidos, y un estudio más exhaustivo puede encontrarse en [Omo94].

Los elementos hardware que se necesitan para los algoritmos que se van a describir consisten sencillamente en un sumador/restador como el visto anteriormente en esta misma sección, un biestable de desbordamiento (v) y tres registros: un acumulador (A), un registro de ampliación del acumulador (Q), y un registro temporal de datos (R), interconectados como se indica en la Figura 9.5.

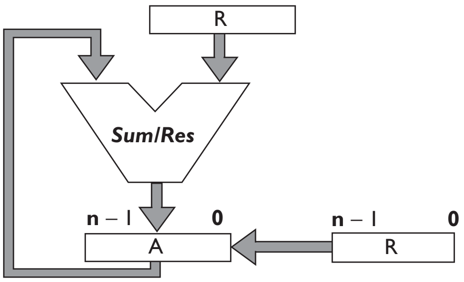


Figura 9.5. *Arquitectura para realizar un multiplicador y un divisor de números sin signo.*

El algoritmo que se va a describir realiza la multiplicación de forma similar a como la realizamos manualmente. En efecto, consiste en realizar una serie de sumas del multiplicando desplazándolo sucesivamente, y de acuerdo con los valores que tiene cada cifra individual del multiplicador, conforme se indica en el organigrama de la Figura 9.6. Inicialmente se carga el multiplicando en R, el multiplicador en Q, y el resultado se obtiene en A (al principio del algoritmo A se inicializa a 0). Puede comprenderse fácilmente el funcionamiento del algoritmo con el ejemplo que se da a continuación.

EJEMPLO 9.6

Suponiendo datos sin signo de n = 4 bits, efectuar el producto 9 × 5, utilizando el algoritmo de la Figura 9.6.
Los números en binario son: $9_{10} = 1001_2$; $5_{10} = 0101_2$.
Las operaciones que se realizan en cada iteracción se muestran en la Tabla 9.1.

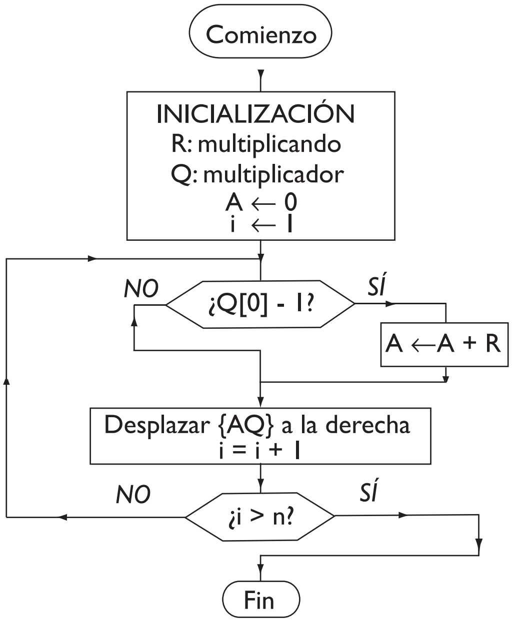


Figura 9.6. *Algoritmo que realiza la multiplicación secuencial de números binarios sin signo.*

Tabla 9.1. *Pasos seguidos por un multiplicador binario secuencial para realizar el producto 9 x 5 .*

Iteracción		**R**	**A**	**Q**
	Carga inicial	**multiplicando** **1001**	**0000**	**multiplicador** **0101**
1	Q[0] = 1 ⇒ A ← A + R Desplazar AQ a dcha.	1001 1001	1001 0100	0101 1010
2	Q[0] = 0 Desplazar AQ a dcha.	1001	0010	0101
3	Q[0] = 1 ⇒ A ← A + R Desplazar AQ a dcha.	1001 1001	1011 0101	0101 1010
4	Q[0] = 0 desplazar AQ a dcha.	1101	0010	1001
				producto $00101101_2 = 45_{10}$

El algoritmo descrito es para números sin signo, en la sección anterior (Sección 9.1.1) indicamos cómo obtener los bits de signo y de desbordamiento, caso de operar con datos con signo, en las distintas representaciones de datos enteros.

Una versión muy mejorada del algoritmo anterior, y válido para números con signo, es el algoritmo de Booth [Omo94, Raf90]. Este algoritmo añade inicialmente un bit 0 a la derecha de Q, y en cada iteracción inspecciona los dos bits menos significativos que se van obteniendo en Q; dependiendo de su valor, suma (si son 01) o resta (si son 10) al acumulador (A) el valor del

multiplicando *R*. Existe otro tipo de multiplicadores, ***multiplicadores combinacionales***, que son mucho más rápidos, pero utilizan un hardware más complejo [Omo94, Raf90].

9.1.2.3 División secuencial para enteros sin signo (con restauración)

Para ilustrar el tipo de operaciones que deben realizarse para implementar la división de números enteros, con el hardware de la Figura 9.5, vamos a considerar un sencillo algoritmo, basado en el método que seguimos para dividir manualmente.

El algoritmo se muestra en la Figura 9.7. Ahora inicialmente se carga el divisor en *R,* el dividendo en *Q,* y *A* se inicializa con 0. El cociente se obtiene en *Q* y el resto en *A.* Puede comprenderse fácilmente el funcionamiento del algoritmo con el ejemplo que se da a continuación.

Existen divisores más rápidos que el anterior, que evitan, por ejemplo, tener que efectuar la restauración. Incluso hay unidades aritméticas que la división de dos números la efectúan obteniendo el inverso del divisor y multiplicando luego por el dividendo. Para más detalles pueden verse, por ejemplo, las referencias [Omo94, Raf90, Pat00].

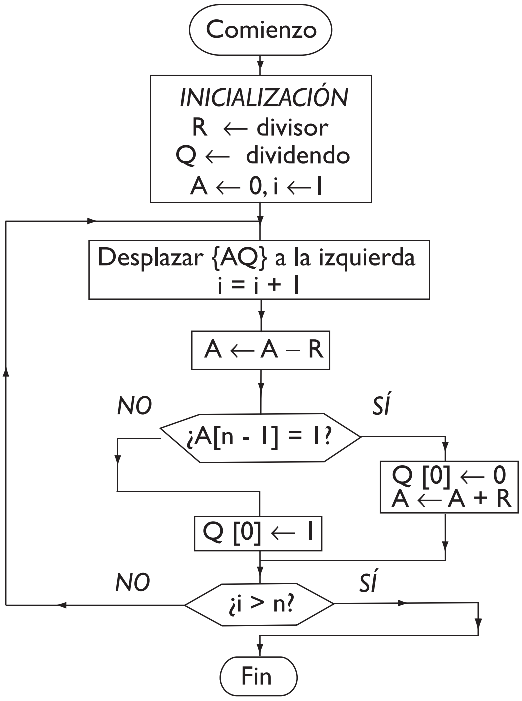


Figura 9.7. *Algoritmo que realiza la división con restauración de números binarios sin signo.*

EJEMPLO 9.7

Suponiendo datos sin signo de n = 4 bits, efectuar la división de 13/5, utilizando el algoritmo de la Figura 9.7.

Los números en binario son: $13_{10} = 1101_2$*;* $5_{10} = 0101_2$*. El resultado que debemos obtener es: A = cociente* $= 2_{10} = 0010_2$*, Q = resto* $= 3_{10} = 0011_2$*.*

Las operaciones que se realizan en cada iteracción se muestran en la Tabla 9.2.

Tabla 9.2. *Pasos seguidos por un multiplicador binario secuencial para realizar el producto 9 x 5.*

Iteración		R	A	Q
	Carga inicial	**divisor** 0101	**0000**	**dividendo** **1101**
1	Desplazar AQ a izquierda	0101	0001	101-
	A ← A – R	0101	1100	101-
	A[3] = 1 ⇒ A ← A + R, q[0] ← 0	0101	0001	101**0**
2	Desplazar AQ a izquierda	0101	0011	010-
	A ← A – R	0101	1110	010-
	A[3] = 1 ⇒ A ← A + R, q[0] ← 0	0101	0011	010**0**
3	Desplazar AQ a izquierda	0101	0110	100-
	A ← A – R	0101	0001	100-
	A[3] = 0 ⇒ Q[0] ← 1	0101	0001	100**1**
4	Desplazar AQ a izquierda	0101	0011	001-
	A ← A – R	0101	1110	001-
	A[3] = 1 ⇒ Q[0] ← 0; A ← A + R	0101	0011	001**0**
			resto 3_{10}	**cociente** 2_{10}

9.1.3 Operaciones básicas con datos en coma flotante

Según se indicó en la Sección 3.6.2, los números reales se almacenan en el computador según una notación exponencial. En toda esta sección nos vamos a referir al sistema de representación interna definido por el estándar IEEE.754 (Sección 3.6.2.1). No detallaremos las operaciones relacionadas con el redondeo, que se realizarán en todos aquellos pasos que impliquen despreciar las cifras menos significativas del número.

Sean los números:

$$N1 = M1 \cdot B^{E1} = [1 \cdot m1] \cdot B^{E1} \quad N2 = M2 \cdot B^{E2} = [1 \cdot m2] \cdot B^{E2}$$

almacenados en la forma:

$$s1\ e1\ m1 \qquad s2\ e2\ m2$$

donde:

$$s = 0 \text{ si } N > 0$$

$$s = 1 \text{ si } N < 0$$

y

$$e1 = S + E1$$

$$e2 = S + E2$$

con

$$S = 2^p - 1, \text{ siendo } p = n_e - 1$$

n_e es el número de bits reservados para almacenar el exponente.

Antes de estudiar las operaciones básicas (suma, resta, multiplicación y división) conviene que repasemos o definamos dos funciones que se utilizan en dichas operaciones básicas.

Normalización

Según vimos, la normalización consiste en reajustar la mantisa y el exponente de forma que se verifique:

$$1 \leq M < 2$$

Como regla práctica podemos decir que para normalizar números representados en binario natural se reajustan el exponente y la mantisa de forma que el uno más significativo de la mantisa ocupe la posición de peso 0 y, en su caso, se efectúa un redondeo.

Igualar exponentes (ecualizar)

Igualar exponentes quiere decir reajustar los exponentes y mantisas de forma que los exponentes sean iguales. Si los números a los que se pretende igualar sus exponentes están normalizados, para hacerlo (y no perder las cifras más significativas) hay que *aumentar el exponente menor*, desplazar la mantisa a la derecha, y efectuar la operación de redondeo correspondiente.

Para estudiar las operaciones básicas entre dos números N_1 y N_2 vamos a denominar al resultado:

$$Nr = Mr \cdot 2^{Er} = [1 . mr] \cdot 2^{Er}$$

que quedará almacenado como:

$$sr\ er\ mr$$

con lo que el valor del número resulta ser:

$$Nr = (-1)^s \cdot [1 . mr] \cdot 2^{er - S}$$

A continuación, analizamos cómo el computador realiza las operaciones básicas en notación exponencial.

Suma (+) $\quad N_r \leftarrow N_1 + N_2$

1) *Igualar los exponentes* de N1 y N2; es decir:
   - $si\ e1 = e2 \quad hacer \quad er' \leftarrow e1; \quad M1' \leftarrow M1; \quad M2' \leftarrow M2$
   - $si\ e1 > e2 \quad hacer \quad er' \leftarrow e1; \quad M1' \leftarrow M1; \quad M2' \leftarrow M2 \cdot 2^{e1-e2}$; *redondear M2'*
   - $si\ e1 < e2 \quad hacer \quad er' \leftarrow e2; \quad M1' \leftarrow M1 \cdot 2^{e2-e1}$; *redondear M1'*; $\quad M2' \leftarrow M2$
2) *Sumar las mantisas* (como números enteros en signo y magnitud): $N_r \leftarrow N_1 + N_2$
   - $si\ s1 = s2; \quad hacer \quad Mr' \leftarrow M1' + M2', \quad sr \leftarrow s_1$

   (observar que la suma se hace con M y no con m; es decir, hay que incluir el bit oculto)
   - $si\ s1 \neq s2$
     - $si\ M1' > M2' \quad hacer \quad Mr' \leftarrow M1' - M2' \quad s_r \leftarrow s_1$
     - $si\ M1' < M2' \quad hacer \quad Mr' \leftarrow M2' - M1' \quad s_r \leftarrow s_2$
3) *Normalizar el resultado, si es necesario; es decir,*
   - *mientras sea* $Mr' > 2 \quad$ *hacer* $\quad Mr \leftarrow Mr' \cdot 2^{-1}$, *redondeo de Mr*; $er \leftarrow er' + 1;$
   - *mientras sea* $Mr' < 1 \quad$ *hacer* $\quad Mr \leftarrow Mr' \cdot 2; \ er \leftarrow er - 1$
4) El número resultante es: *sr er mr.*

Nota: En cualquiera de las fases anteriores:

- si $M1' = 0$ hacer $N_r = N_2$ y acabar;
- si $M2' = 0$; hacer $N_r = N_1$, y acabar; o si hay desbordamiento hacer $v = 1$ y acabar.

Resta (–) $N_r \leftarrow N_1 - N_2$

1) *Igualar los exponentes* de N1 y N2; es decir:
   si e1 = e2 hacer er' ← e1; M1' ← M1; M2' ← M2
   si e1 > e2 hacer er' ← e1; M1' ← M1; M2' ← M2 · 2^{e1-e2}; redondear M2'
   si e1 < e2 hacer er' ← e2; M1' ← M1 · 2^{e2-e1}; redondear M1'; M2' ← M2
2) *Restar las mantisas* (como números enteros en signo y magnitud): $N_r \leftarrow N_1 - N_2$
   si $s_1 = s_2$
   si M1' > M2' hacer Mr' ← M1' – M2' sr ← s1
   si M1' < M2' hacer Mr' ← M2' – M1' sr
   si s1 ≠ s2; hacer Mr' ← M1' + M2', sr ← s1

Las restas indicadas pueden efectuarse sumando al minuendo el complemento a dos del substraendo.

3) *Normalizar el resultado, si es necesario; es decir,*
   - *mientras sea Mr' > 2 hacer Mr ← Mr' · 2^{-1}, redondeo de Mr; er ← er' + 1;*
   - *mientras sea Mr' < 1 hacer Mr ← Mr' · 2 ; er ← er–1*
4) El número resultante es: *sr er mr.*

Nota: En cualquiera de las fases anteriores:
- si *M1' = 0* hacer *Nr = N2* y acabar;
- si *M2' = 0;* hacer *Nr = N1* y acabar; o
- si hay desbordamiento hacer *v = 1* y acabar.

Multiplicación $N_r \leftarrow N_1 \cdot N_2$

1) *Exponentes:*
   Sumar los exponentes:
   $$Er \leftarrow E1 + E2 \qquad\qquad er \leftarrow e1 + e2 - S$$
   No hay que olvidar que los exponentes almacenados son los exponentes originales "exceso S", por lo que a la suma de los exponentes de los operandos almacenados hay que restarle S.
2) *Mantisas:*
   Multiplicar las mantisas, como números binarios sin signo (Sección 9.1.2):
   $$Mr \leftarrow M1 \cdot M2$$
3) *Signo:* si los signos de los operandos son iguales el signo del resultado es positivo, y si son distintos es negativo; es decir:
   $$s_r \leftarrow s_1 \oplus s_2$$
4) *Normalizar,* si es necesario, el resultado. Es decir,
   - *mientras sea Mr'> 2, hacer: Mr ← Mr' · 2^{-1}, redondeo de Mr, y er ← er'+ 1;*
   - *mientras sea Mr' < 1 hacer: Mr ← Mr' · 2 ; er ← er–1*
5) *El número resultante es: sr er mr*

Nota: En cualquiera de las fases anteriores:
- si *M1' = 0* hacer *Nr = 0* y acabar;
- si *M2' = 0*; hacer *Nr = N0*, y acabar; o
- si hay desbordamiento hacer *v = 1* y acabar.

División $N_r \leftarrow N_1/N_2$

1) *Exponentes:*

   Restar los exponentes:

$$Er \leftarrow E1 - E2 \qquad er \leftarrow e1 - e2 + S$$

   No hay que olvidar que los exponentes almacenados son los exponentes originales "exceso S", por lo que a la resta de los exponentes de los operandos almacenados hay que sumarles el sesgo, S.

2) *Mantisas:*

   Dividir las mantisas, como números binarios sin signo (Sección 9.1.2):

$$M_r \leftarrow \frac{M1}{M2}$$

3) *Signo:* si los signos de los operandos son iguales el signo del resultado es positivo, y si son distintos es negativo; es decir:

$$s_r \leftarrow s_1 \oplus s_2$$

4) *Normalizar,* si es necesario, el resultado. Es decir,
   - *mientras sea Mr'> 2,* *hacer:* $Mr \leftarrow Mr' \cdot 2^{-1}$*, redondeo de Mr, y* $er \leftarrow er' + 1$;
   - *mientras sea Mr' < 1* *hacer:* $Mr \leftarrow Mr' \cdot 2$; $er \leftarrow er^{-1}$

5) El número resultante es: *sr er mr*

Nota: En cualquiera de las fases anteriores:

- si $M1' = 0$ y $M2' = 0$; hacer $e = 2^{ne} - 1$ y $m \neq 0$; es decir, el resultado es indefinido: $Nr = NaN$;
- si $M1' = 0$ hacer $Nr = 0$ y acabar;
- si $M2' = 0$; hacer $e = 2^{ne} - 1$ y $m = 0$; es decir, se representa $Nr = +\infty$, si $sr = 0$, o $Nr = -\infty$, si $sr = 1$;
- si hay desbordamiento hacer $v = 1$ y acabar.

9.1.4 Implementación de la unidad de coma flotante

Hasta principios de la década de los años 90 la mayoría de procesadores no disponían de circuitos específicos para realizar las operaciones en coma flotante. En estos casos el software del sistema disponía de rutinas que implementaban los algoritmos anteriormente descritos, lo que ralentizaba considerablemente al computador sobre todo en aplicaciones de tipo científico o técnico que requieren mucho cálculo. Para resolver el problema se desarrollaron **coprocesadores aritméticos**, que son chips que se conectan al mismo bus del procesador central, y que son capaces de interpretar las instrucciones que se captan de memoria a través del bus, ejecutando las instrucciones que le son propias: operaciones en coma flotante, incluyendo funciones exponenciales, logarítmicas y trigonométricas. El coprocesador contiene, por tanto, circuitos de control, registros para datos, circuitos aritméticos, etc. Intel desarrolló coprocesadores aritméticos para sus microprocesadores para computadores personales (PC) hasta el 80486 (excepto su versión SX). Los microprocesadores modernos incluyen en su interior una **unidad de coma flotante** (**FPU**, *Floating Point Unit*) con los circuitos típicos de los coprocesadores aritméticos, notablemente mejorados.

Tabla 9.3. *Coprocesadores aritméticos utilizables en diversos microprocesadores.*

Microprocesador	Coprocesador asociado
8086	8087
8088	8087
80286	80287
68000	68881
80386SX	80387SX
80386DX	80387DX
80486SX	80387SX
80486DX	*interno*
Pentium	*interno*

9.2 Unidad de tratamiento

Como se indicó al inicio de este capítulo, vamos a diseñar aquí el ordenador didáctico elemental (CODE-2) a nivel de micromáquina. En esta sección diseñaremos el camino de datos (o unidad de tratamiento), y en la sección siguiente (Sección 9.3) la unidad de control.

El **nivel de micromáquina**, como cualquier otro nivel, utiliza como elementos constructivos los módulos proporcionados por el nivel inmediato inferior, en este caso el nivel de lógica digital, estudiado en los Capítulos 5 y 6. En el caso concreto de CODE, los módulos que utilizaremos para diseñar su camino de datos se incluyen en la Figura 9.8. En la última columna de la figura se especifica la sección de los Capítulos 5 o 6 donde se ha estudiado y diseñado el elemento correspondiente.

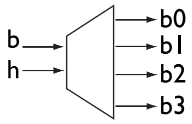
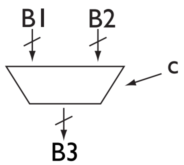
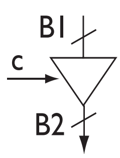
Elemento	**Esquema**	**Descripción**	**Diseño a nivel de lógica digital**
Decodificador binario		Todas las salidas son 0, excepto la que corresponde al valor decimal de la entrada. Si bh = 00, b0 = 1; si bh = 01, b1 = 1; si bh = 10, b2 = 1; y si bh = 11, b3 = 1	*Secc. 5.4.5*
Selector o multiplexor		En el bus de salida (B3) aparece la información de uno y sólo uno de los buses de entrada; así, en el caso de buses de 16 bits, si c = 0, B3(15:0) = B1(15:0), y si c = 1, B3(15:0) = B2(15:0)	*Secc. 5.4.6*
Adaptador triestado		Si c = 0, la salida está en estado de alta impedancia, pudiendo el bus B2 estar eléctricamente conectado a la salida de otro circuito. Si c = 1, la información del bus B1 aparece en el bus B2	*Secc. 5.4.7*

Figura 9.8. *Módulos del nivel de lógica digital utilizados para diseñar CODE-2 a nivel de micromáquina.*

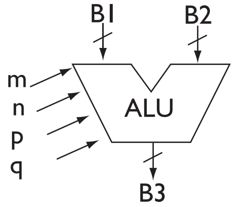
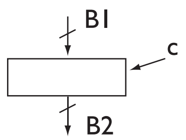
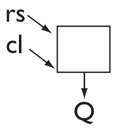
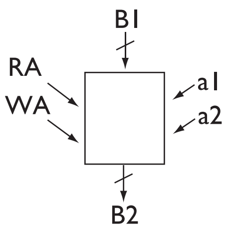
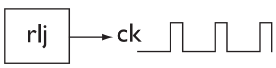
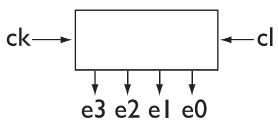
Elemento	Esquema	Descripción	Diseño a nivel de lógica digital
ALU		En B3 se obtiene el resultado de efectuar una operación entre los datos de los buses B1 y B2. La operación concreta de salida se determina con las señales de control m, n, p y q	*Secc. 5.4.10*
Registro		Cuando c = 1, la información del bus de entrada (B1) se graba en el registro, apareciendo siempre en el bus de salida (B2) la información almacenada (16 bits, caso de buses de 16 bits)	*Secc. 6.3*
Biestable asíncrono		Si rs = 1, queda almacenado en el biestable un 1; y si cl = 1, queda almacenado un 0.	*Secc. 6.2.1*
Banco de registros (16 registros de 16 bits)		RA se compone de 4 hilos, que seleccionan el registro a leer; WA se compone también de 4 hilos y selecciona el registro a grabar. Si a1 = 1, aparece en el bus de salida (B2) la información del registro seleccionado con RA; y si a2 = 1, se escribe en el registro seleccionado por WA la información que exista en el bus de entrada (B1). *Nota*: simultáneamente se puede leer y escribir	*Secc. 6.3.1*
Reloj		Circuito que genera una secuencia periódica e indefinida de pulsos (ck), de periodo T y frecuencia F = 1/T	*Secc. 1.2*
Contador binario de 4 bits		Circuito que cuando cl = 1, pone a 0 los 4 bits almacenados (e3:e0), y al llegar un pulso ck, incrementa en 1 su valor; es decir, según llegan los pulsos ck, se genera la secuencia 0000, 0001, 0010, 0011, 0100, ..., 1110, 1111	*Secc. 6.4*

Figura 9.8. *Módulos del nivel de lógica digital utilizados para diseñar CODE-2 a nivel de micromáquina. (Cont.)*

La estructura de CODE-2 se muestra en la Figura 9.9, y puede observarse que coincide en gran medida con la del computador visto en el Capítulo 7 (Figuras 7.9 y 7.10). En el caso de CODE-2, como ya indicamos en el Capitulo 8, el registro *rE* hace las funciones de puntero de pila (*SP*). Obsérvese que la Figura 7.9 sólo contenía los elementos de CODE *directamente* utilizables por el programador de lenguaje máquina y ensamblador. En la Figura 9.9 se han añadido el registro de instrucción (*IR*), el contador de programa (*PC*), y los registros de datos (*DR*) y de direcciones (*AR*).

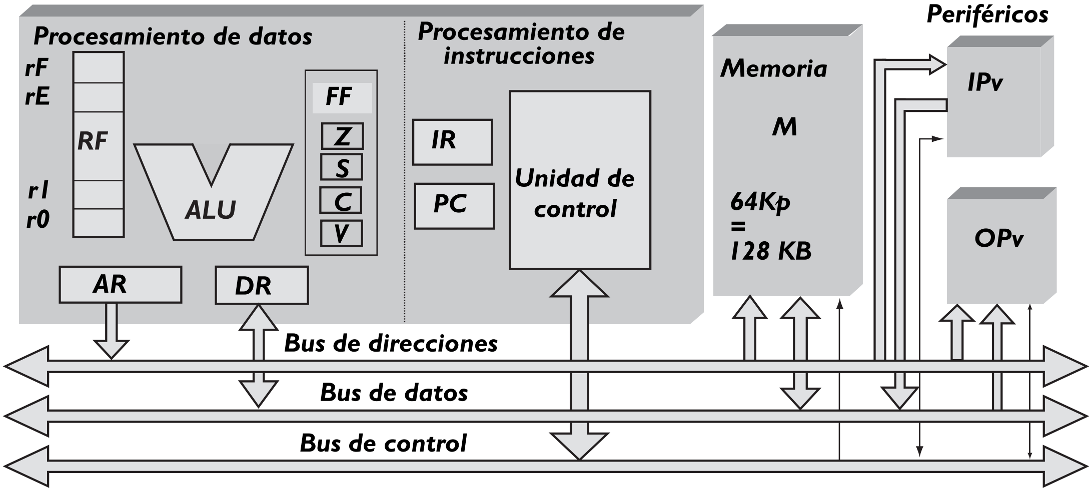


Figura 9.9. *Elementos básicos de CODE-2.*

La Figura 9.10 muestra un esquema completo de CODE-2 a nivel de micromáquina, en donde se especifican las interconexiones entre todos sus elementos. En esta sección nos vamos a centrar en el camino de datos. Las señales indicadas con letras minúsculas y dirigidas a los distintos elementos a través de una flecha representan señales de control, también denominadas **microórdenes**, que se describirán en detalle cuando se analice la unidad de control (Sección 9.3).

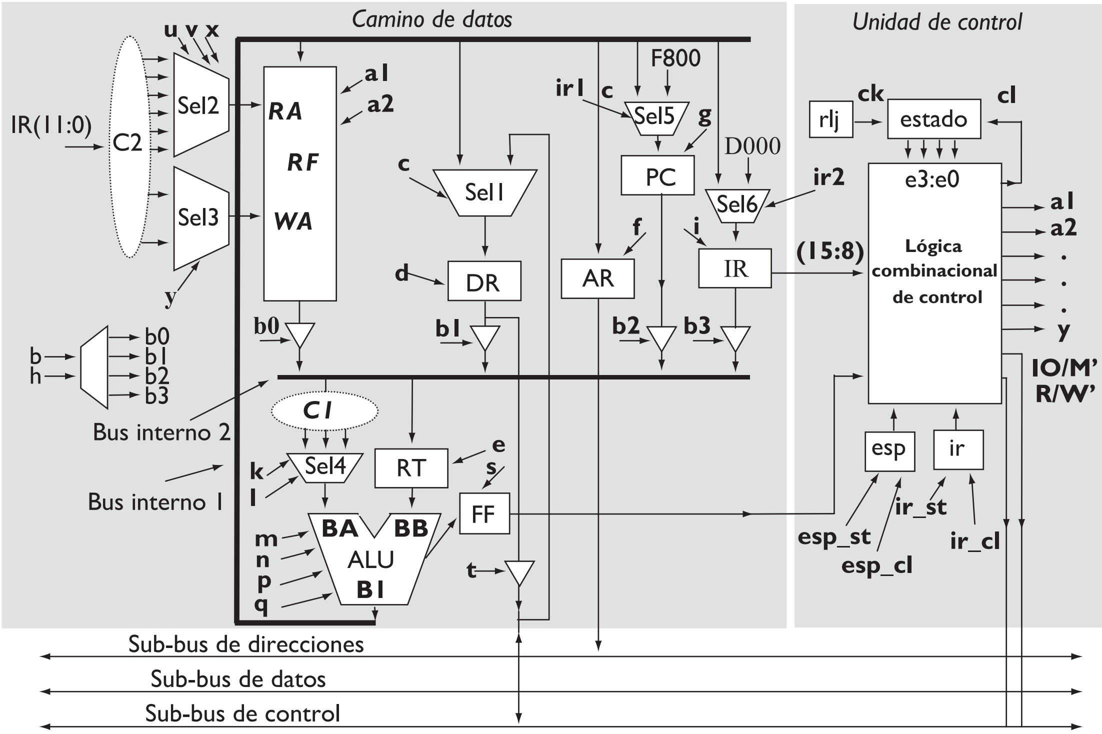


Figura 9.10. *Esquema completo del procesador CODE-2.*

La comunicación entre los distintos elementos se efectúa a través de dos buses internos:

- **Bus interno 1,** (abreviadamente **Bus1**): a través de él se puede conectar la salida de la *ALU* (*B1*) con las entradas del banco de 16 registros (*RF*), las entradas de los registros *DR*, *AR* e *IR*, y la entrada del contador de programa *PC*. Para llevar la información de *B1* a un registro concreto:

  (a) las señales de control de su selector de entrada (caso de tenerlo) deben conectar el bus a la entrada de dicho registro, y

  (b) debe darse un pulso en la señal de control de carga del registro.

  Así, por ejemplo, para cargar la información a la salida de la *ALU* en el contador de programa, la señal *ir1* de *Sel5* debe conectar el bus a la entrada de *PC*, y dar un pulso de carga en paralelo de *PC* ($g = 1$). Obviamente, como muestra la figura, todas las señales de control (en este caso *ir1* y *g*) son generadas por la unidad de control.
- **Bus interno 2, (Bus2):** Conecta las salidas de *RF*, *DR*, *AR* e *IR* con las entradas de la *ALU*. Las señales de control *b0, b1, b2* y *b3* de los correspondientes circuitos tri-estado determinan cuál de dichos registros deposita su contenido en el *Bus2*. Si, por ejemplo, hubiese que llevar el contenido de *DR* al registro temporal, *RT,* de la ALU, la unidad de control tendría que hacer $b1 = 1$ para que la salida de *DR* pase al estado de baja impedancia, y dar un pulso de carga en *RT* ($e = 1$). Obsérvese que únicamente una de las 4 señales citadas anteriormente puede estar a 1 (ya que sino se produciría un cortocircuito en el *Bus1*), por lo que, por seguridad, dichas señales se generan con un decodificador binario 2×4, a partir de las señales de control *b* y *h* (incluido en la parte izquierda de la figura).

Puede observarse que todos los elementos del camino de datos están descritos en la Figura 9.8, a la que antes nos hemos referido. Algunas aclaraciones sobre la estructura son:

1) El módulo *C1* (que aparece en la figura en forma circular) representa interconexiones que se realizan entre los hilos del *Bus2* y las entradas al selector *Sel4*, concretamente:
   - La entrada 0 del selector está formada por los 16 hilos del bus: *Bus2(15:0).*
   - La entrada 1 del selector está formada por 8 bits cero y los 8 bits menos significativos de *Bus2;* es decir: *H'00##Bus(7:00)* (El operador ## significa concatenar).
   - La entrada 2 del selector es igual a la de entrada 1, pero con los bytes intercambiados; es decir: *Bus(7:00)##H'00.*
2) El módulo *C2* (que aparece en la figura en forma circular) representa las conexiones que se efectúan para las entradas de los selectores Sel2 y Sel3 de la siguiente manera:
   - Conexión de las salidas *IR(3:0)* del registro instrucción (*IR*) a la entrada 0 de *Sel2.*
   - Conexión de las salidas *IR(7:4)* del registro instrucción (*IR*) a la entrada 1 de *Sel2.*
   - Conexión de las salidas *IR(11:8)* del registro instrucción (*IR*) a la entrada 2 de *Sel2.*
   - Conexión del valor *H'D* (es decir, *1101*) a la entrada 3 de *Sel2.*
   - Conexión del valor *H'E* (es decir, *1110*) a la entrada 4 de *Sel2.*
   - Conexión de las salidas *IR(11:8)* del registro instrucción (*IR*) a la entrada 0 de *Sel3.*
   - Conexión del valor *H'F* (es decir, *1110*) a la entrada 1 de *Sel3.*
3) El banco de registros (*RF*) contiene 16 registros de 16 bits. La entrada *RA* (de 4 bits) selecciona el registro a leer, y la entrada *WA* (4 bits) el registro a grabar, con información procedente del Bus1. Las señales de control asociadas a *RF* son:
   - $a1 = 1$, grabar el contenido de bus1 en el registro especificado en *WA.*

- *a2 = 1*, leer el contenido del registro especificado por *RA*.
- *b0 = 1*, las salida de *RF* se lleva al *bus2*.

4) La salida del registro de datos (*DR*), puede conectarse al bus2, o al bus de datos externo. Las señales de control asociadas a *DR* son:
   - *d*: carga en paralelo de *DR*.
   - *b1 = 1*; el contenido de *DR* pasa al *bus2*.
   - *t = 1*; el contenido de *DR* pasa al bus de datos.
5) El registro de dirección (*AR*) tiene como entrada *bus1*, y como salida el bus de direcciones externo. La señal de control asociadas a *AR* es:
   - *f*: carga en paralelo de *AR*.
6) El contador de programa es sencillamente un registro (similar a los demás), ya que los incrementos en uno que tienen que realizarse en las fases de captación de instrucción se realizan en la ALU. Las señales de control asociadas con él son:
   - *g*: carga en paralelo de *PC*.
   - *b2 = 1*; la salida de *PC* se lleva al bus2.
7) El registro de instrucción *IR* tiene asociadas las siguientes señales de control:
   - *i*: carga en paralelo de *IR*.
   - *b3 = 1*; la salida de *IR* se lleva al *bus2*.
8) El registro temporal, *RT,* de la ALU tiene asociada la siguiente señal de control:
   - *e*: carga en paralelo de *RT*.
9) La *ALU* tiene en una de sus entradas un registro temporal, *RT,* de forma que las operaciones que realiza las hace con el dato que le llega del *Bus interno 1* (a través de las conexiones *C1* y del selector *Sel4*) y con el contenido de *RT*. Las operaciones que se realizan en la ALU son las que se indican en la Tabla 9.4.
10) Los biestables indicadores (*FF*) se activan con la señal s:
    - *s*: señal de carga de los biestables *FF*, de acuerdo con el resultado de la *ALU.*

Tabla 9.4. *Operaciones que se realizan en la ALU y sus señales de control.*

Señales mnpq	Operaciones en la ALU (B1)
0000	BA
0001	BA + 1
0010	BA – 1
0011	BB
0100	BB + 1
0101	BB – 1
0110	Suma de BA y BB (complemento a 2)
0111	Resta BA-BB (complemento a 2)
1000	NAND; es decir, (BA · BB)’
1001	Desplazamiento a izquierda
1010	Desplazamiento a derecha
1011	Desplazamiento aritmético a derecha

Los selectores actúan de la siguiente forma:

- **Sel1:** Selecciona la entrada del registro DR:
  - Si *c = 0; DR ← bus1*
  - Si *c = 1; DR ← bus de datos* (externo)
- **Sel2:** Selecciona el código que especifica el registro de RF a leer, de acuerdo con la Tabla 9.5.
- **Sel3:** Selecciona el código que especifica el registro *RF* a grabar:
  - *Si y = 0, WA ← IR(11:8)*
  - *Si y = 1, WA ← H'F*
- **Sel4:** Selecciona la entrada BA de la ALU de la siguiente manera (Tabla 9.6)
  - *Si k = 0, l = 0; BA ← Bus2(15:0),*
  - *Si k = 0, l = 1; BA ← H'00##Bus(7:00),*

  V*Si k = 1, l = 0; BA ← Bus(7:00)##H'00*
- **Sel5:** Selecciona la entrada del contador de programa (*PC*):
  - *Si ir1 = 0; PC ← bus1*
  - *Si ir1 = 1; PC ← H'F800*. (*H'F800* es la dirección de memoria ROM donde se inicia la rutina de atención de interrupciones, se cargará en el contador de programa en el momento de atender una interrupción).
- **Sel6:** Selecciona la entrada del registro instrucción (IR):
  - *Si ir2 = 0; IR ← bus1*
  - *Si ir1 = 1; DR ← H'D000* (H'D000 es la instrucción de llamada a subrutina, que se cargará en el registro instrucción para atender una petición de interrupción).

Tabla 9.5. *Salidas proporcionadas por las conexiones C2 y el selector Sel2.*

Señales uvx	Operaciones en el selector de registro a leer (Sel2)	Explicación
000	RA ← IR(3:0)	Leer el registro indicado en el campo ra
001	RA ← IR(7:4)	Leer el registro indicado en el campo rs
010	RA ← IR(11:8)	Leer el registro indicado en el campo rx
011	RA ← XH'D	Leer el registro rD (dirección de salto)
100	RA ← XH´E	Leer el registrro rE (puntero de pila)

Tabla 9.6. *Salidas proporcionadas por las conexiones C1 y el selector Sel4.*

Señales ki	Operaciones en el selector de entrada en la ALU (Sel4)	Explicación
00	alu(BA) ← bus2(15:0)	Se lleva a la entrada BA de la ALU los 16 bits del Bus2
01	alu(BA) ← 00'H##bus2(7:0)	Se lleva a BA los 8 MSBs, y los más significativos se hacen 0
10	alu(BA) ← Bus2(7:0)##00'H	Se cambian los 8 LSBs por los MSBs, y los LSBs se hacen 0
11		

En la Tabla 9.7 se incluyen las 30 señales de control de todos los elementos de CODE-2. Las señales *cl* y *esp* e *ir*, son propias de la unidad de control y se explicarán más adelante.

Recuérdese (Sección 7.1) que las señales $IO/\overline{M}$ e $R/\overline{W}$ son para controlar, a través del bus de control del sistema, las transferencias entre el procesador y la memoria o los distintos puertos de E/S. Actuando adecuadamente sobre el conjunto de señales de control se puede llevar una información de un sitio a otro, realizar cualquier operación de la ALU, etc.; en definitiva, controlar el funcionamiento del computador.

Tabla 9.7. *Señales de control del camino de datos de CODE-2.*

Señales	Operaciones	Explicación
a1	a1 = 1, adaptador B0 bus2 ← RF	Selección de lectura del banco de registros (RF): leer el registro RA de RF
a2	a2 = 1, RF ← bus1	Selección de escritura en RF: almacenar en el registro WA de RF
bh	bh = 00 (b0 = 1), bus2 ← RF	Selección del daptador triestado b0
	bh = 01 (b1 = 1), bus2 ← DR	Selección del daptador triestado b1
	bh = 10 (b2 = 1), bus2 ← PC	Selección del daptador triestado b2
	bh = 11 (b3 = 1), bus2 ← IR	Selección del daptador triestado b3
c	c = 0, DR ← bus1; c = 1, DR ← bus datos	
cl	poner a 0 contador de estado	Para provocar el inicio de la captación de una nueva instrucción (estado c0)
d	d = 1, cargar DR	
e	e = 1, cargar RT	
esp_rs	esp_rs = 1, pone a 1 biestable esp	Para entrar en espera en el próximo estado C0
f	f = 1, carga de AR	
g	g = 1, carga de PC	
i	i = 1, cargar IR	
ir1	ir1 = 0, PC ← bus1; ir1 = 1, PC ← H'F800	Dirección de salto de interrupción
ir2	ir2 = 0, IR ← bus1; ir2 = 1, IR ← H'D000	Provocar llamada a subrutina
ir_cl	ir_cl = 1, pone a 0 biestable ir	Final de ciclo de interrupción
kl	Sel4	Reordenación de bytes del Bus2 (IR), véase Tabla 9.6
mnpq	ALU	Selección de función a realizar por la ALU, véase Tabla 9.4
s	carga FF	Para activar biestables de estado de la ALU (sólo en determinadas instrucciones)
t	t = 1; bus de datos ← DR	
uvx	Sel2	Selección de registro de RF a leer, ver Tabla 9.5
y	y = 0, WA ← rx; y = 1, WA ← F'H	Selección de registro de RF a escribir

9.3 Unidad de control

Según vimos en la Sección 1.2, el objetivo de la unidad de control es monitorizar el funcionamiento de todo el computador dirigiendo la información a las unidades o elementos precisos en el momento oportuno y dando las órdenes adecuadas para que se realicen las acciones pertinentes (memorizar, sumar, etc.). Todo ello lo hace la unidad de control, analizando o *interpretando* (Figura 9.10) la instrucción máquina (almacenada en *IR*), y los valores de los biestables indicadores (*FF*), biestable de espera (*esp)* y biestable de petición de interrupción (*ir)*. En función de este análisis genera las 29 señales de control (*a1, a2, b, c, ..., y, cl, ir_cl, ir_cl, esp_st,* $D/\overline{M}$, y $R/\overline{W}$) que gobiernan el funcionamiento de los distintos elementos del computador. En definitiva, podemos decir que la unidad de control capta las instrucciones, las interpreta, capta los operandos y

los lleva a las unidades funcionales adecuadas de la unidad de procesamiento, envía las señales pertinentes a la ALU para completar las operaciones implicadas en la instrucción y a otros elementos para que los resultados se almacenan en el destino correspondiente.

La unidad de control realiza las funciones descritas en el párrafo anterior generando señales de control (**microórdenes**) que producen determinadas **microoperaciones** (Sección 7.3) en un orden predeterminado, de forma sincronizada con un **estado máquina**.

Se pueden utilizar dos metodologías para diseñar la unidad de control, y que se analizan a continuación. La primera de ellas se denomina de **lógica cableada** (en inglés: *hardwired*) (Sección 9.3.1) y se basa en la utilización de puertas lógicas, o circuitos PLA o FPGA, etc. y de las técnicas de diseño de sistemas combinacionales y secuenciales vistas en los Capítulos 5 y 6. La segunda se denomina **microprogramación**, (Sección 9.3.2) y se basa en almacenar en una memoria, por lo general de sólo lectura (**memoria de control**), las microórdenes que implementan las distintas instrucciones: la ejecución de una instrucción sencillamente implica leer de la memoria de control la secuencia de las microórdenes correspondientes a ella.

La forma más adecuada de comprender los fundamentos de las dos metodologías indicadas es considerar un ejemplo. Obviamente, en el contexto de esta obra, el mejor ejemplo es CODE-2.

9.3.1 Unidad de control cableada de CODE-2

Tal como se describió en la Sección 7.3, la unidad de control de CODE-2 debe ejecutar continua y automáticamente un ciclo que consta de dos fases: captación de instrucción y ejecución. La unidad de control se diseña para ejecutar el repertorio de instrucciones de la Tabla 8.4, que se resume en la Tabla 9.7 a la que se le ha añadido una primera columna en la que se asigna un rótulo (*I0* a *IF*) que identifica a cada una de las instrucciones, para poderlas referenciar más cómodamente.

Tabla 9.8. *Resumen de las instrucciones máquina de CODE-2.*

Identificación	Codop		Nombre	Explicación	Nº ciclos
	binario	Hex			
I0	0000	0	Cargar	rx ← M(rD + v)	9
I1	0001	1	Almacenar	M(rD + v) ← rx	9
I2	0010	2	Carga inmediata baja	rx(15:8) ← H'00; rx(7:0) ← v	6
I3	0011	3	Carga inmediata alta	rx(15:8) ← v	8
I4	0100	4	Entrada	rx ← IPv	8
I5	0101	5	Salida	OPv ← rx	8
I6	0110	6	Suma	rx ← rs + ra	7
I7	0111	7	Resta	rx ← rs – ra	7
I8	1000	8	NAND	rx ← (rs · ra)'	7
I9	1001	9	Desplaza izquierda	C ← rx(15), rx(i) ← rx(i – 1), i = 15, …, 1; rx(0) ← 0	6
IA	1010	A	Desplaza derecha	C ← rx(0), rx(i) ← rx(i + 1), i = 0, …, 14; rx(15) ← 0	6
IB	1011	B	Desplaza arit. dcha.	C ← rx(0), rx(i) ← rx(I + 1), i = 0, …, 14	6
IC	1100	C	Salto	Si cnd se cumple, PC ← rD	6
ID	1101	D	Subrutina	Si cnd se cumple, rE ← rE – 1, M(rE) ← PC, PC ← rD	6/9
IE	1110	E	Retorno	PC ← M(rE); rE ← rE + 1	8
IF	1111	F	Parar	Parar	6

9.3.1.1 Ciclo de la unidad de control

Según vimos con detalle en la Sección 7.3, la ejecución de cada instrucción consume varios ciclos de reloj (estados), los primeros para captar la instrucción y los últimos para la ejecución propiamente dicha de la instrucción. En el caso de CODE-2, antes de iniciar la ejecución de cualquier instrucción se puede entrar en un estado de parada o espera indefinida (Sección 8.3.3). Este estado de espera es controlado por un biestable, cuya salida denominamos *esp*, que se pone a uno (para entrar en espera) cuando la señal de control *esp_st* se hace uno; y se hace cero con una señal *esp_cl* generada a través del teclado de CODE-2 (véase Sección 9.6). Después se realiza la captación de la instrucción, que puede ser la que se encuentre en memoria, apuntada por el contador de programa (*PC*), o una llamada (*D000*) a una rutina de gestión de interrupciones grabada en la memoria ROM a partir de la dirección H'F800. Si el biestable de petición de interrupción *ir* (de *interruption request*) está a *1* se hace la llamada a la rutina de gestión de interrupciones citada. Posteriormente, como se indica en la Figura 9.11, se tiene la fase de ejecución de instrucción. Obsérvese que cada instrucción se identificada con el rótulo indicado en la primera columna de la Tabla 9.8.

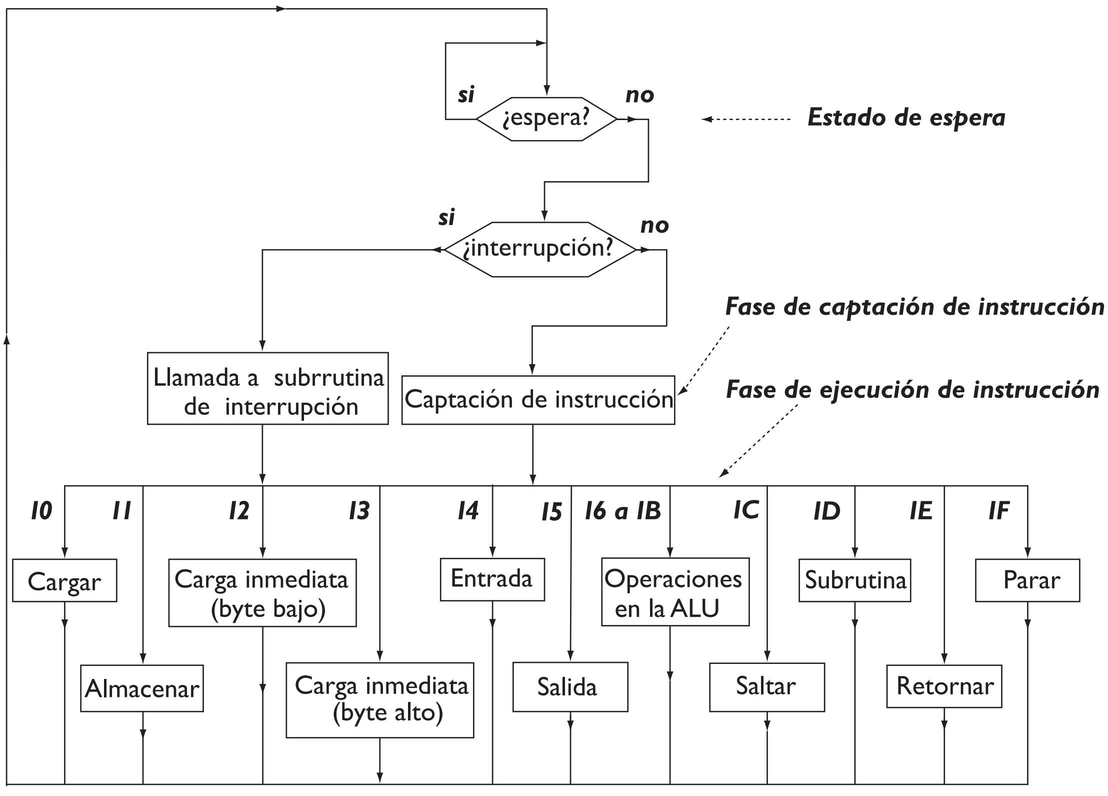


Figura 9.11. *Esquema simplificado del ciclo de la unidad de control de CODE-2.*

La Figura 9.12 incluye un diagrama más detallado, que muestra las microoperaciones que deben generarse para implementar las distintas instrucciones del repertorio de instrucciones máquina de la unidad de control. En el caso de CODE-2 (véase la Figura 9.12) los ciclos de reloj o **estados máquina** los simbolizamos como *c0, c1, ..., c9.*

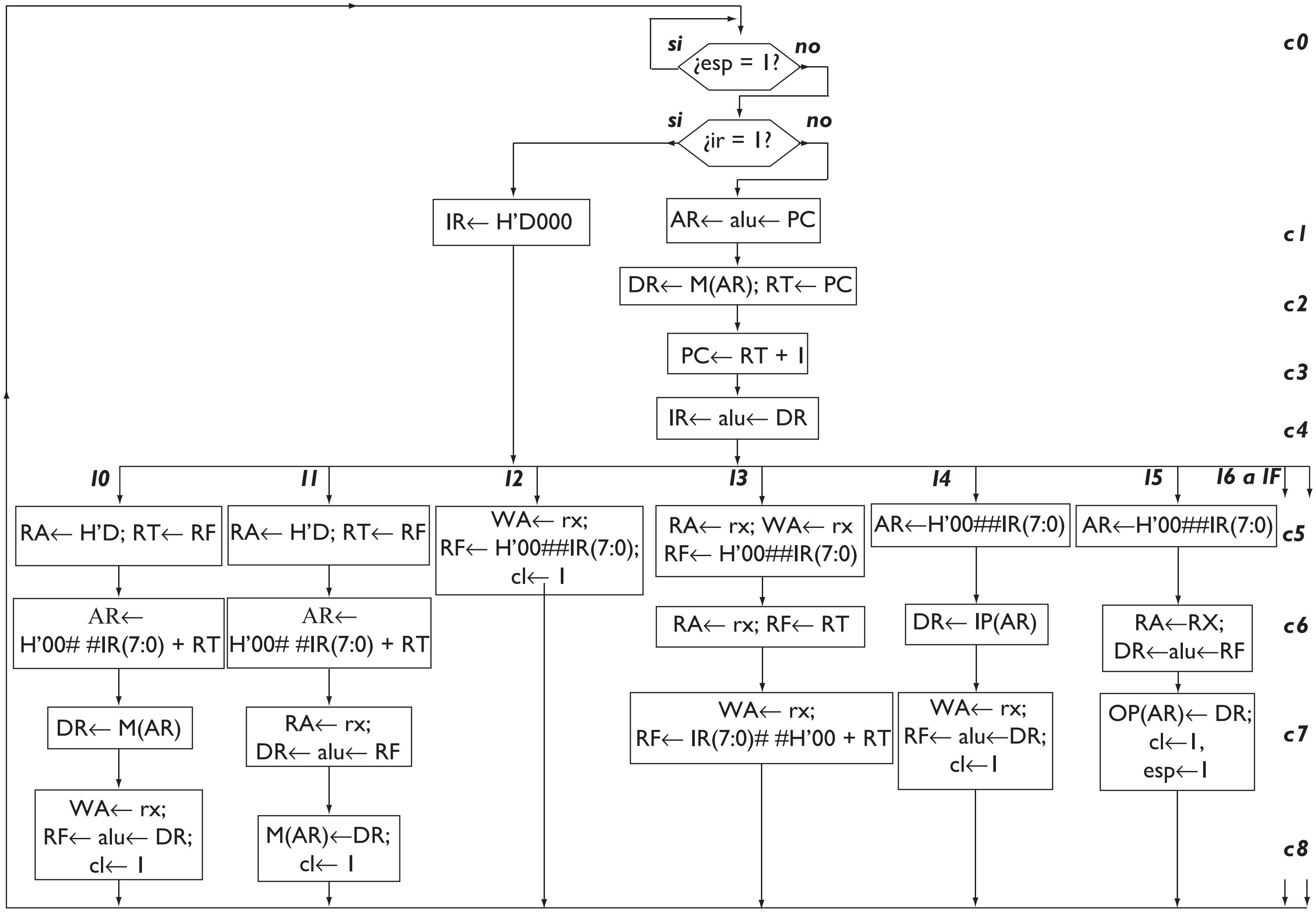


Figura 9.12. *Diagrama de flujo del repertorio de instrucciones de CODE-2.*

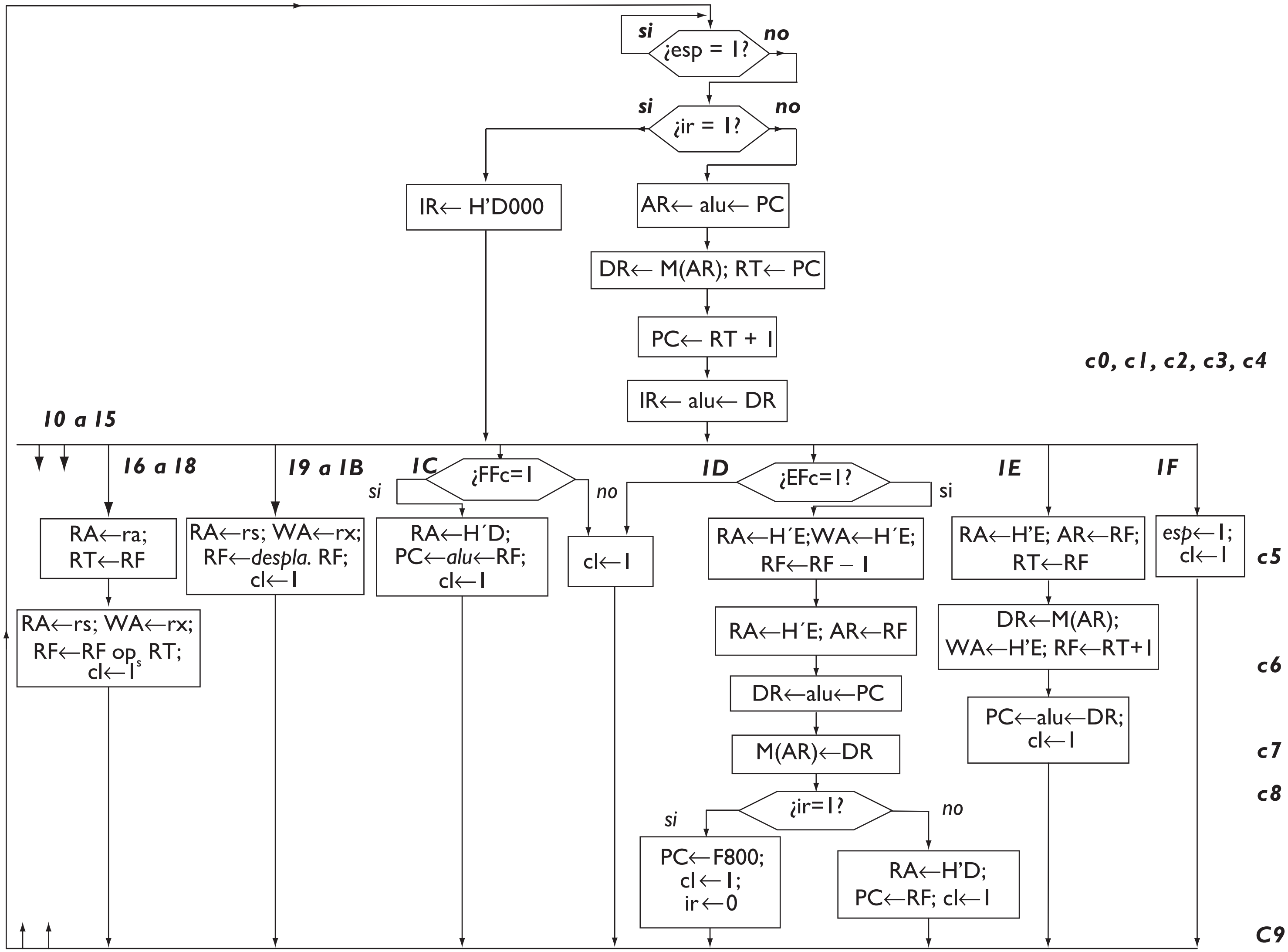


Figura 9.12. *Diagrama de flujo del repertorio de instrucciones de CODE-2.(Cont.)*

Después de detectar si el biestable de espera está a uno (*esp* = *1*), y de si hay una petición de interrupción (*ir* = *1)*, y caso de que no se hayan producido ninguna de estas dos eventualidades, se efectúa la captación de instrucción. Para ello (véase Figura 9.10) se lleva el contenido del contador de programa al registro del bus de dirección ($AR \leftarrow alu \leftarrow PC$), se hace la lectura de memoria ($DR \leftarrow M[AR]$), se suma uno al contador de programa (cosa que se realiza en la ALU: $RT \leftarrow PC; PC \leftarrow RT + 1$) y por último se lleva la instrucción leída al registro instrucción ($IR \leftarrow alu \leftarrow DR$). En la Figura 9.12 se indican cómo se secuencian estas operaciones en los estados *c1, c2, c3* y *c4*. La estrategia a seguir para realizar el diagrama de flujo es:

a) secuenciar adecuadamente las microoperaciones, y
b) incluir en el mismo estado las microoperaciones que sean compatibles, de forma que se consuma el menor número posible de ciclos (cosa que se hace, por ejemplo en el estado *c2* con las microoperaciones $DR \leftarrow M(AR)$ y $RT \leftarrow PC$).

A continuación, como la instrucción a ejecutar ya se encuentra en el registro instrucción (*IR*), puede pasarse a la ejecución propiamente dicha de la instrucción. Para seguir obteniendo el diagrama de control hay que recordar la función que realiza cada instrucción (quinta columna de la Tabla 9.8), considerar el esquema de la Figura 9.12, y seguir la estrategia indicada en el párrafo anterior. A titulo de ejemplo consideremos la instrucción *I0*; es decir *LD rx,[rD + v],* que carga un registro con un contenido de memoria. Recuérdese que la función que realiza esta instrucción es: $rx \leftarrow M(rD + v);$ donde, el valor de *v*, a sumar a *rD* se encuentra en *IR(7:0).* Para ejecutar la instrucción, primero habrá que sumar *rD* con *v,* luego leer de la memoria el dato, y por último llevar el dato al registro *rx* del banco de registros (*RF*). Las microoperaciones serán las siguientes:

Estado c5

Llevar el contenido del registro *rD* de *RF* al registro temporal de la *ALU*; es decir:

$$RA \leftarrow H'D; RT \leftarrow RF$$

Estado c6

Sumar el valor de v (al que habrá que anteponer 8 ceros, por ser la *ALU* de 16 bits) el valor de *rD*, que se encuentra ya en *RT*, y llevarlo al registro de dirección:

$$AR \leftarrow H'00\#\#IR(7:0) + RT$$

Estado c7

Como ya se tiene la dirección del dato en *AR*, puede efectuarse la lectura de memoria:

$$DR \leftarrow M(AR)$$

Estado c8

El dato ya se encuentra en el registro de datos (*DR*), ahora hay que llevarlo (*escribirlo)* en el registro *rx* del banco de registros, para lo cual habrá que dar la dirección de *rx* en la entrada *WA* de *RF*, para especificar la dirección del registro a grabar, llevar el contenido al Bus1, que está conectado a la entrada del banco de registros, y dar la orden de escribir en éste último. Todo ello lo simbolizamos con las siguientes microoperaciones:

$$WA \leftarrow rx; RF \leftarrow alu \leftarrow DR$$

Por último, en este mismo estado, se puede generar una microorden que indica a la propia unidad de control que ha finalizado la ejecución de la instrucción; esta microorden, que explicaremos más adelante, es:

$$cl \leftarrow 1$$

De esta forma hemos completado las microoperaciones de la instrucción *I0*, y que aparecen en la primera rama de la izquierda de la Figura 9.12. De la misma forma se pueden ir estableciendo las microoperaciones para el resto de las instrucciones y completando el diagrama de flujo. Conviene indicar que en la figura *FFc* representa la salida del biestable que impone la condición de bifurcación (salto o llamada a subrutina); y siempre es uno cuando la bifurcación en incondicional.

9.3.1.2 Microoperaciones a generar por la unidad de control

Analizando la exposición realizada anteriormente y la Figura 9.12, podemos obtener las distintas microoperaciones que controlan el funcionamiento de CODE-2, incluidas por orden alfabético en la primera columna de la Tabla 7.9. Nos hemos limitado a llevarlas a dicha columna sin que aparezca ninguna repetida. En total hay 40 microoperaciones. En la Tabla 9.9 y sucesivas, con objeto de facilitar la composición de las mismas, denotamos el complemento de una variable con un apóstrofo, por ejemplo (RF · RT)', en lugar de con una raya en su parte superior, $\overline{RF} \cdot \overline{RT}$.

Tabla 9.9. *Lista de las microoperaciones de CODE-2, de las señales de control que las implementan, y de la situación en que debe generarse cada una de ellas.*

Microoperaciones	Señales a generar	Situación en que debe generarse la microoperación
AR ← H'00##IR(7:0)	b = h = 1, k = 0,l = 1, m = n = p = q = 0, f = 1	c5 · I4 + c5 · I5
AR ← 00'H##IR(7:0) + RT	b = h = 1, k = m = 0, n = p = 1, q = 0, f = 1	c6 · I0 + c6 · I1
AR ← alu ← PC	b = 1, h = k = j = m = n = p = q = 0, f = 1	c1 · ir'
AR ← RF	a1 = 1, b = h = 0, k = 0, I = 0, f = 1	c6 · ID · FFc + c5 · IE
cl ← 1	cl = 1	c8 · I0 + c8 · I1 + c5 · I2 + c7 · I3 + c7 · I4 + c7 · I5 + c6 · I6 + c6 · I7 + c6 · I8 + c5 · I9 + c5 · IA + c5 · IB + c5 · IC + c9 · ID + c7 · IE + c5 · IF
DR ← alu ← PC	b = 1, h = 0, m = n = p = q = 0, c = 0, d = 1	c7 · ID · FFc + c7 · IE
DR ← alu ← RF	a1 = 1, b = h = 0, m = n = p = q = 0, c = 0, d = 1	c6 · I5
DR ← IP(AR)	IO/M' = 1, R/W' = 1, c = 1, d = 1	c6 · I4
DR ← M(AR);	IO/M' = 0, R/W' = 1, c = 1, d = 1	c2 · ir' + c7 · I0 + c6 · IE
esp ← 1	esp_st = 1	c7 · I5 + c5 · IF
IR ← alu ← DR	b = 0, h = 1, m = n = p = q = 0, ir2 = 0, i = 1	c4 · ir'
IR ← H'D000	ir2 = 1, i = 1	c1 · ir
ir ← 0	ir_cl = 1	c9 · ID · FFc · ir
M(AR) ← DR	t = 1, IO/M' = 0, R/W' = 0	c8 · I1 + c8 · ID · FFc
OP(AR) ← DR	t = 1, IO/M' = 1, R/W' = 0	c7 · I5
PC ← alu ← DR	b = 0, h = 1, m = n = p = q = 0, ir1 = 1, g = 1	c7 · IE
PC ← alu ← RF	a1 = 1, b = h = 0, m = n = p = q = 0, ir1 = 0, g = 1	c9 · ID · FFc · ir'

Tabla 9.9. *Lista de las microoperaciones de CODE-2, de las señales de control que las implementan, y de la situación en que debe generarse cada una de ellas. (Cont.)*

Microoperaciones	Señales a generar	Situación en que debe generarse la microoperación
PC ← H'F800	ir1 = 1, g = 1	c9 · ID · FFc · ir
PC ← RT + 1	b = 1, h = 0, k = 0, l = 0, m = p = q = 0, n = 1, ir1 = 0, g = 1	c3 · ir'
RA ← D'H	u = 0, v = 1, x = 1	c5 · I0 + c5 · I1 + c5 · IC · FFc + c9 · ID · FFc · ir'
RA ← E'H	u = 1, v = 0, x = 0	c5 · ID · FFc + c6 · ID · FFc + c5 · IE
RA ← ra	u = 0, v = 0, x = 0	c5 · (I6 + I7 + I8)
RA ← rs	u = 0, v = 0, x = 1	c6 · (I6 + I7 + I8)
RA ← rx	u = 0, v = 1, x = 0	c7 · I1 + c5 · I3 + c6 · I3 + c6 · I5 + c5 · (I9 + IA + IB)
RF ← H'00##IR(7:0)	b = h = 1, k = 0, l = 1, m = n = p = q = 0, a2 = 1	c5 · I2
RF ← H'00##RF(7:0)	a1 = 1, b = h = 0, k = 0, l = 1, m = n = p = q = 0, a2 = 1	c5 · I3
RF ← alu ← DR	b = 0, h = 1, k = l = 0, m = n = p = q = 0, a2 = 1	c8 · I0 + c7 · I4
RF ← *despla. izda.* RF	a1 = 1, b = h = 0, k = l = 0, n = p = 0, m = q = 1, s = 1, a2 = 1	c5 · I9
RF ← *despla. dcha.* RF	a1 = 1, b = h = 0, k = l = 0, n = q = 0, m = p = 1, s = 1, a2 = 1	c5 · IA
RF ← *despla. arit. dcha.* RF	a1 = 1, b = h = 0, k = l = 0, n = 0, m = p = q = 1, s = 1, a2 = 1	c5 · IB
RF ← IR(7:0)##00'H + RT	b = h = 1, k = 1, l = 0, n = q = 0, n = p = 1, a2 = 1	c7 · I3
RF ← RF – 1	a1 = 1, b = h = 0, k = l = 0, m = n = q = 0, p = 1, s = 1, a2 = 1	c5 · ID · FFc
RF ← RF + RT	a1 = 1,b = h = 0, k = l = 0, n = p = 1, m = q = 0, s = 1, a2 = 1	c6 · I6
RF ← RF – RT	a1 = 1, b = h = 0, k = l = 0, m = 0, n = p = q = 1, s = 1, a2 = 1	c6 · I7
RF ← (RF · RT)' *(nand)*	a1 = 1, b = h = 0, k = l = 0, n = p = q = 0, m = 1, s = 1, a2 = 1	c6 · I8
RF ← RT + 1	m = p = q = 0, n = 1, a2 = 1	c6 · IE
RT ← PC	b = 1, h = 0, e = 1	c2 · ir'
RT ← RF	a1 = 1, b = h = 0, e = 1	c5 · (I0 + I1 + I6 + I7 + I8 + IE) + c6 · I3
WA ← E'H	y = 1	c5 · ID · FFc + c6 · IE
WA ← rx	y = 0	c5 · (I2 + I3 + I9 + IA + IB) + c6(I6 + I7 + I8) + c7 · (I3 + I4) + c8 · I0

Cada microoperación se implementa por medio de microórdenes (señales de control), que fácilmente podemos establecer consultando la Figura 9.10 y la Tabla 9.7. Por ejemplo, para implementar la microoperación:

$$AR \leftarrow alu \leftarrow PC$$

habrá que:

1) Hacer $b2 = 1$ (es decir, $b = 1, h = 0$), para que el contenido de *PC* llegue a *Bus2* (Tabla 9.7).
2) Hacer $k = 0$ y $j = 0$, para que el contenido del Bus2 llegue sin ninguna modificación a la entrada *BA* de la *ALU* (Tabla 9.6).
3) Hacer $m = n = p = q = 0,$ para que la entrada *BA* de la ALU aparezca en *Bus1* (Tabla 9.4).
4) Cargar el contenido del *Bus1* en *AR*; es decir, generar un pulso $f = 1$ (Tabla 9.7).

En resumen, si la unidad de control genera las siguientes microórdenes:

$$b = 1,\ m = n = p = q = 0,\ f = 1$$

se realizarán las acciones ligadas a la microoperación indicada. Estas microordenes las hemos escrito en la segunda columna de la Tabla 9.9, en la fila que corresponde a la microoperación $AR \leftarrow alu \leftarrow PC$ (tercera fila). De igual forma podemos analizar el resto de las microoperaciones y completar la columna indicada. Por otra parte nos interesa establecer las circunstancias en que cada microoperación debe ser generada. Para ello vamos a utilizar la columna tercera de la Tabla 9.9, y suponer que las señales generadas por la unidad de control son siempre cero, salvo que se especifique lo contrario.

Con ayuda del diagrama de flujo del repertorio (Figura 9.12) observamos que la microoperación que estamos utilizando como ejemplo ($AR \leftarrow alu \leftarrow PC$), debe ser producida en los ciclos c1 (es decir, cuando c1 = 1) caso de que ir = 0; analíticamente expresamos esta condición como $c1 \cdot \overline{ir}$, y la llevamos a la fila correspondiente de la Tabla 9.9. En resumen, la unidad de control debe hacer $b = 1$ y $f = 1$, cuando se verifique que $c1 \cdot \overline{ir}$; esto analíticamente se expresa así:

$$b = c1 \cdot \overline{ir}\ ;\ f = c1 \cdot \overline{ir}$$

y significa: hacer *b* igual a uno si *c1* es uno y simultáneamente *ir* es cero; hacer *f* igual a uno si *c1* es uno y simultáneamente *ir* es cero; que es lo que se pretendía.

El mismo razonamiento lo podemos hacer para las otras microoperaciones. Veamos otros dos casos más.

Para implementar la microinstrucción:

$$DR \leftarrow M(AR)$$

Habrá que (Figura 9.10 y Tabla 9.7):

1) Hacer $IO/\overline{M} = 0$, para especificar a las unidades externas al procesador que la operación a realizar a través del bus del sistema (bus externo al procesador) es con la memoria, y por tanto el contenido de *AR* es una dirección de memoria (Tabla 9.7); y $R/\overline{W} = 1$ para especificar *lectura (de memoria)*. Estas dos señales de control, que constituyen dos hilos del bus de control, provocan que el contenido de la posición *AR* de memoria se lea y aparezca en el bus de datos.
2) Hacer $c = 1$, para que el contenido del bus de datos aparezca a la entrada del registro *DR*, y generar un pulso para cargar el registro *DR*; es decir, hacer $d = 1.$

En resumen, si la unidad de control genera las siguientes microórdenes:

$$IO/\overline{M} = 0,\ R/\overline{W} = 1,\ c = 1,\ d = 1$$

se realizarán las acciones ligadas a la microoperación indicada (*DR* ← *M(AR)*).

Para establecer las circunstancias en que la microoperación debe ser generada, examinamos el diagrama de microoperaciones (Figura 9.12) y observamos que la microoperación *DR* ← *M(AR)* debe producirse en los ciclos *c2* (es decir, cuando *c2 = 1*) caso de que *ir = 0*; analíticamente expresamos esta condición como $c2 \cdot \overline{ir}$, o cuando se ejecute la instrucción LD (es decir, cuando *I0 = 1*) en los ciclos *c7* (es decir, cuando *c7 = 1*), o en las instrucciones IE (RET, retorno) en el ciclo *c6*. En resumen, la unidad de control debe generar la microinstrucción indicada cuando sea 1 la siguiente expresión:

$$c2 \cdot \overline{ir} + c7 \cdot IO + c6 \cdot IE$$

que significa los siguiente: *"implementar la microinstrucción DR ← M(AR) cuando c2 e ir sean 0, o cuando c7 e I0 sean igual a 1, o cuando c6 e IE sean 1"*. Obsérvese que, como se vio con detalle en el Capítulo 5, el operador " · " representa el *Y lógico* y el operador " + " el *O lógico.*

Como último ejemplo consideremos la microinstrucción *cl* ← *1.* De la Figura 9.12 se deduce fácilmente que debe generarse cuando se verifica:

$$c8 \cdot I0 + c8 \cdot I1 + c5 \cdot I2 + c7 \cdot I3 + c7 \cdot I4 + c7 \cdot I5 + c6 \cdot I6 + c6 \cdot I7 + c6 \cdot I8 + c5 \cdot I9 + c5 \cdot IA + c5 \cdot IB + c5 \cdot IC + c9 \cdot ID + c7 \cdot IE + c5 \cdot IF$$

que significa lo siguiente: *"generar la microinstrucción cl ← 1 cuando c7 = 1 e I0 = 1, o cuando c7 = 1 e I1 = 1, ... , o cuando c5 = 1 e IF = 1";* como se indica en la tercera columna de la Tabla 9.8, en la fila correspondiente a la microinstrucción indicada.

9.3.1.3 Implementación de la unidad de control cableada

La unidad de control, como se indica en la Figura 9.13, tiene que generar las distintas microórdenes (*cl, a1, a2, b, ...* $IO/\overline{M}$, $R/\overline{W}$), en función de las siguientes señales que actúan como entradas de la **lógica combinacional de control**:

- estado, definido en forma codificada por los bits *e3, e2, e1, e0;* a partir de los cuales se obtiene el estado máquina decodificado: *c0, c1,..., c9;*
- código de operación de la instrucción en curso; que, teniendo en cuenta el formato de las instrucciones máquina, está contenido en los bits *IR(15:12)* del registro instrucción;
- salidas de los biestables de espera (*es*) y de petición de interrupción (*ir*), y
- en las instrucciones de salto y llamadas a subrutinas, de la condición de bifurcación, que, teniendo en cuenta el formato de las instrucciones, está contenida en los bits *IR(11:8)* del registro de instrucción.

En la parte superior de la Figura 9.13 se encuentra el reloj, que sencillamente es un circuito que genera una secuencia continua de pulsos, y puede diseñarse con circuitos integrados específicos, tales como el temporizador *555* o el generador *8284* [Flo06]. La señal de reloj, *ck,* actúa sobre el **registro de estado** que en el caso de CODE-2 es un contador binario de 4 bits (Figura 9.8). Según llegan los pulsos *ck* en las salidas del contador *(e3:e0)* va apareciendo la siguiente secuencia, que identifica en forma codificada cada uno de los estados máquina (entre paréntesis): *0000 (c0), 0001 (c1), 0010 (c2), 011 (c3), ..., 1111 (c15), 0000 (c1), 0001 (c2)....* Por otra parte, el contador dispone de una entrada, *cl,* de puesta a cero; con ella se consigue que en el último estado de cada instrucción (Figura 9.12) se interrumpa la secuencia y se pase directamente al estado *c0* de inicio de la siguiente instrucción. Así, por ejemplo, el último estado de la instrucción *LD* (*I0* en el diagrama de

flujo de la Figura 9.12) es el *c8 (1000),* y cuando se llega a él se genera la señal *cl* = *0,* que provoca que el siguiente estado sea el *c0 (0000),* en lugar del *c9 (1001)* y sucesivos.

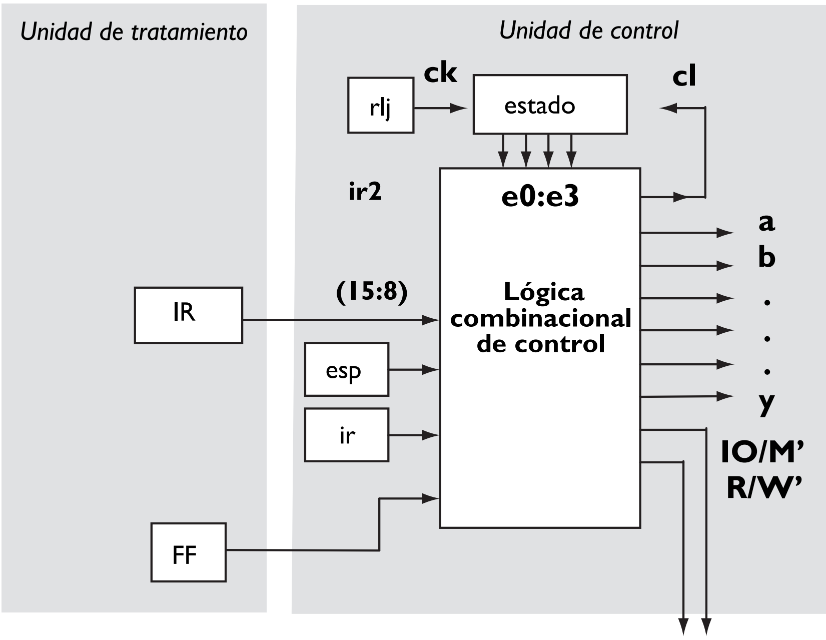


Figura 9.13. *Esquema de la unidad de control.*

Pero, analicemos con más detalle el interior de la lógica combinacional de control, con ayuda del esquema que se muestra en la Figura 9.14. Para facilitar la generación de las señales de control utilizamos dos decodificadores binarios (Figura 9.8) de 4 entradas a 16 salidas y un selector de 8 entradas:

- Un **decodificador de estado**, que genera las señales que genera los estados decodificados *c0, c1, …c9* a partir de las salidas, e3:e0, del contador de estado. Cuando *e3:e0 = 0000* las salidas del decodificador son: *c0 = 1, c2 = c3 = ... = c15 = 0*; después *e3:e0 = 0001*, y se hace *c1 = 1, c0 = c2 = ... = c15 = 0*; a continuación *e3:e0 = 0010* y las salidas del decodificador son: c2 = 1, y c0 = c1 = c3 = ... = c15 = 0; y así sucesivamente.
- Un **decodificador de instrucción**, que genera las señales *I0, ...IF* que identifican individualmente a cada instrucción; obviamente estas señales se obtienen directamente del código de operación de la instrucción que se encuentra en el registro de instrucción (IR). Con este decodificador cuando *IR(15:12) = 0000* las salidas del decodificador son: *I0 = 1, I1 = I2 = ... = IF = 0*; si *IR(15:12) = 0001* las salidas del decodificador son: *I1 = 1, I0 = I2 = I3 = ... = IF = 0*; etc. En definitiva, si en el registro instrucción se encuentra la instrucción *x,* la variable *Ix* es uno y las restantes son cero.
- Un **selector de condición**, que selecciona el biestable de condición (*Z, S, C, V*) especificado por el campo formado por los bits 11:8 de la instrucción máquina, y que se encuentra en las salidas *IR11:IR8* del registro instrucción. El valor del biestable que establece la condición se obtiene en la salida *FFc.* Obsérvese que cuando el salto en incondicional, *IR(11:8)* será *0000*, y entonces *FFc = 1*, con lo que en este caso siempre se producirá el salto, como debe ser.

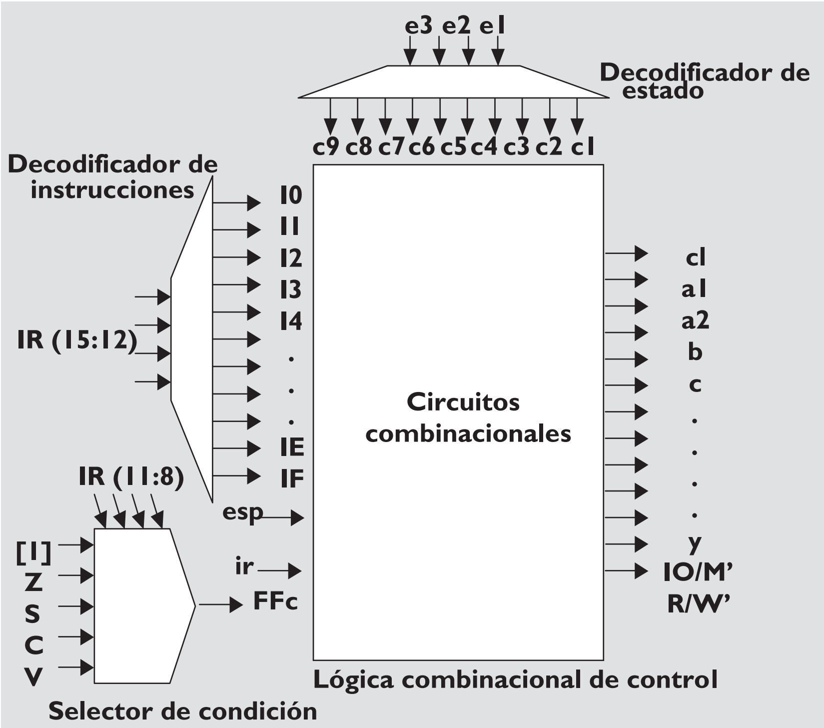


Figura 9.14. *Esquema simplificado de la lógica combinacional de la unidad de control.*

Ahora, como claramente queda puesto de manifiesto en la Figura 9.14, el problema queda reducido a generar las microórdenes *cl, a1, a2, b, ...* $IO/\overline{M}$, $R/\overline{W}$, en función de las entradas *c0...c9, I0...IF*, y de los bits *FFc* (condición de salto), *esp* (espera) e *ir* (petición de interrupción). Es un problema muy sencillo de resolver aplicando las técnicas descritas en la Sección 5.3, ya que cada microorden puede ser descrita como una función de conmutación en términos de las variables de entrada citadas anteriormente.

Las funciones de conmutación se pueden obtener a partir de la segunda y tercera columna de la Tabla 9.9 de la siguiente forma:

1) Elegimos una microorden a generar, la *c*, por ejemplo.
2) Buscamos en la segunda columna de la tabla donde aparece la microorden con valor uno; así, $c = 1$ aparece en la microoperación:

$$DR \leftarrow IP(AR)$$

3) En la tercera columna se indica en qué *situación* debe tomar la microorden el valor uno, con lo que podemos sin más igualar la variable que identifica a la microorden con la expresión booleana que aparece en la columna tercera. En nuestro ejemplo $c = 1$ si *c6* e *I4* son uno; es decir:

$$c = c6 \cdot I4$$

4) Puede ser que la microorden en análisis tenga que ser uno en otra microoperación o situación; por lo que tendremos que volver al *Paso 2*; apareciendo nuevas expresiones o

términos que deben añadirse a la función de conmutación con un operador " + ". Debe analizarse la tabla hasta su final. En el caso de la microorden *c*; también debe ser uno con la microorden:

$$DR \leftarrow M(AR)$$

que tiene que generarse (tercera columna) cuando se verifique:

$$c2 \cdot ir' + c7 \cdot I0 + c6 \cdot IE$$

por lo que la función de conmutación que sintetiza *a1* es:

$$c = c6 \cdot I4 + c2 \cdot ir' + c7 \cdot I0 + c6 \cdot IE = c2 \cdot ir' + c6 \cdot (I4 + IE) + c7 \cdot I0$$

Obsérvese que la expresión anterior indica analíticamente que *c* debe ser uno en los ciclos c2 cuando ir sea cero, o en los ciclos *c6* cuando *I4* o *IE* sean uno, o en los ciclos *c7* cuando *I0* sea uno.

De esta forma se pueden obtener todas las funciones de conmutación que sintetizan las distintas microórdenes (Tabla 9.10).

Tabla 9.10. *Funciones de conmutación que sintetizan las señales de control de CODE-2.*

a1 = c6 · ID · FFc + c5 · IE + c6 · I5 + c9 · ID · FFc · ir' + c5 · I3 + c5 · I9 + c5 · IA + c5 · IB + c5 · ID · FFc + c6 · I6 + c6 · I7 + c6 · I8 + c5 · (I0 + I1 + I6 + I7 + I8 + IE) + c6 · I3
a2 = c5 · I2 + c5 · I3 + c8 · I0 + c7 · I4 + c5 · I9 + c5 · IA + c5 · IB + c7 · I3 + c5 · ID · FFc + c6 · I6 + c6 · I7 + c6 · I8 + c6 · IE
b = c5 · I4 + c5 · I5 + c6 · I0 + c6 · I1 + c1 · ir' + c7 · ID · FFc + c7 · IE + c7 · ID · FFc + c7 · IE + c3 · ir' + c5 · I2 + c7 · I3 + c2 · ir'
c = c6 · I4 + c2 · ir' + c7 · I0 + c6 · IE
cl = c5 · (I2 + I9 + IA + IB + IC + IF) + c6 · (I6 + I7 + I8) + c7 · (I3 + I4 + I5 + IE) + c8 · (I0 + I1) + c9 · ID
d = c7 · ID · FFc + c7 · IE + c7 · ID · FFc + c7 · IE + c6 · I5 + c6 · I4 + c2 · ir' + c7 · I0 + c6 · IE +
e = c2 · ir' + c5 · (I0 + I1 + I6 + I7 + I8 + IE) + c6 · I3
esp_st = c7 · I5 + c5 · IF
f = c5 · I4 + c5 · I5 + c6 · I0 + c6 · I1 + c1 · ir' + c6 · ID · FFc + c5 · IE
g = c7 · IE + c9 · ID · FFc · ir' + c9 · ID · FFc · ir + c3 · ir'
h = c5 · I4 + c5 · I5 + c6 · I0 + c6 · I1 + c4 · ir' + c7 · IE + c5 · I2 + c8 · I0 + c7 · I4 + c7 · I3
i = c4 · ir' + c1 · ir +
ir1 = c7 · IE + c9 · ID · FFc · ir +
ir2 = c1 · ir
ir_cl = c9 · ID · FFc · ir
k = c7 · I3
l = c5 · I4 + c5 · I5 + c5 · I2 + c5 · I3 +
m = c5 · I9 + c5 · IA + c5 · IB + c6 · I8
n = c6 · I0 + c6 · I1 + c3 · ir' + c7 · I3 + c6 · I6 + c6 · I7 + c6 · IE
p = c6 · I0 + c6 · I1 + c5 · IA + c5 · IB + c7 · I3 + c5 · ID · FFc + c6 · I6 + c6 · I7
q = c5 · I9 + c5 · IB + c6 · I7
s = c5 · I9 + c5 · IA + c5 · IB + c5 · ID · FFc + c6 · I6 + c6 · I7 + c6 · I8
t = c8 · I1 + c8 · ID · FFc + c7 · I5 +
u = c5 · ID · FFc + c6 · ID · FFc + c5 · IE +
v = c5 · I0 + c5 · I1 + c5 · IC · FFc + c9 · ID · FFc · ir' + c7 · I1 + c5 · I3 + c6 · I3 + c6 · I5 + c5 · (I9 + IA + IB) +
x = c5 · I0 + c5 · I1 + c5 · IC · FFc + c9 · ID · FFc · ir' + c6 · (I6 + I7 + I8)
y = c5 · ID · FFc + c6 · IE
IO/M' = c6 · I4 + c7 · I5
R/W' = c6 · I4 + c2 · ir' + c7 · I0 + c6 · IE

Una vez obtenidas las funciones de conmutación, debe decidirse la tecnología a utilizar para su implementación (puertas lógicas, PLA, FPGA, etc.), y aplicar los procedimientos descritos en el Capítulo 5 para transformar y simplificar las funciones en consonancia con la tecnología elegida.

Optemos por una implementación con puertas lógicas. La Tabla 9.11 muestra las funciones de conmutación, minimizadas y utilizando la variable intermedia $IDc = ID \cdot FFc$. En la Figura 9.15, a título de ejemplo, se muestran los circuitos que implementan mediante puertas lógicas las siguientes cuatro microórdenes: *c, k, y, R/W'*. De forma similar, a partir de la Tabla 9.11, se podrían sintetizar el resto de microórdenes que implementan las microoperaciones que monitorizan y controlan el funcionamiento de CODE-2.

Tabla 9.11. *Funciones de conmutación de las señales de control de CODE-2, minimizadas para una implementación AND-OR.*

a1 = c5 · (I0 + I1 + I3 + I6 + I7 + I8 + I9 + + IA + IB + IDc + IE) + c6 · (I3 + I5 + I6 + I7 + I8 + IDc)
a2 = c5 · (I2 + I3 + I9 + IA + IB + IDc) + c6 · (I6 + I7 + I8 + IE) + c7 · (I3 + I4) + c8 · I0
b = c1 · ir'c2 · ir' + c3 · ir' + c5 · (I2 + I4 + I5) + c6(I0 + I1) + c7 · (I3 + IDc + IE)
c = c2 · ir' + c6 · (I4 + IE) + c7 · I0
cl = c5 · (I2 + I9 + IA + IB + IC + IF) + c6 · (I6 + I7 + I8) + c7 · (I3 + I4 + I5 + IE) + c8 · (I0 + I1) + c9 · ID
d = c2 · ir' + c6 · (I5 + I4 + IE) + c7 · (I0 + IDc + IE + c7 · IE)
e = c2 · ir' + c5 · (I0 + I1 + I6 + I7 + I8 + IE) + c6 · I3
esp_st = c5 · IF + c7 · I5
f = c1 · ir' + c5 · (I4 + I5 + IE) + c6 · (I0 + I1 + IDc)
g = c3 · ir' + c7 · IE + c9 · IDc
h = c4 · ir' + c5 · (I2 + I4 + I5) + c6 · (I0 + I1) + c7 · (I3 + I4 + IE) + c8 · I0
i = c1 · ir + c4 · ir'
ir1 = c7 · IE + c9 · IDc · ir
ir2 = c1 · ir
ir_cl = c9 · IDc · ir
k = c7 · I3
l = c5 · (I2 + I3 + I4 + I5)
m = c5 · (I9 + IA + IB) + c6 · I8
n = c3 · ir' + c6 · (I0 + I1 + I6 + I7 + IE) + c7 · I3
p = c5 · (IA + IB + IDc) + c6 · (I0 + I1 + I6 + I7)
q = c5 · (I9 + IB) + c6 · I7
s = c5 · (I9 + IA + IB + IDc) + c6 · (I6 + I7 + I8)
t = c7 · I5 + c8 · (I1 + IDc)
u = c5 · (Idc + IE) + c6 · IDc
v = c5 · (I0 + I1 + I3 + I9 + IA + IB + IC · FFc) + c6 · (I3 + I5) + c7 · I1 + c9 · IDc · ir'
x = c5 · (I0 + I1 + IC · FFc) + c6 · (I6 + I7 + I8) + c9 · IDc · ir'
y = c5 · IDc + c6 · IE
IO/M' = c6 · I4 + c7 · I5
R/W' = c

Recuérdese que, tal y como indicamos en la introducción a este capítulo, existen herramientas que permiten diseñar automáticamente sistemas digitales en general, y procesadores en particular partiendo de una descripción HDL de su repertorio de instrucciones máquina. En la pagina web de este texto hemos incluido un diseño de CODE-2 alternativo al aquí presentado, utilizando una FPGA y partiendo de una descripción VHDL [Diao2].

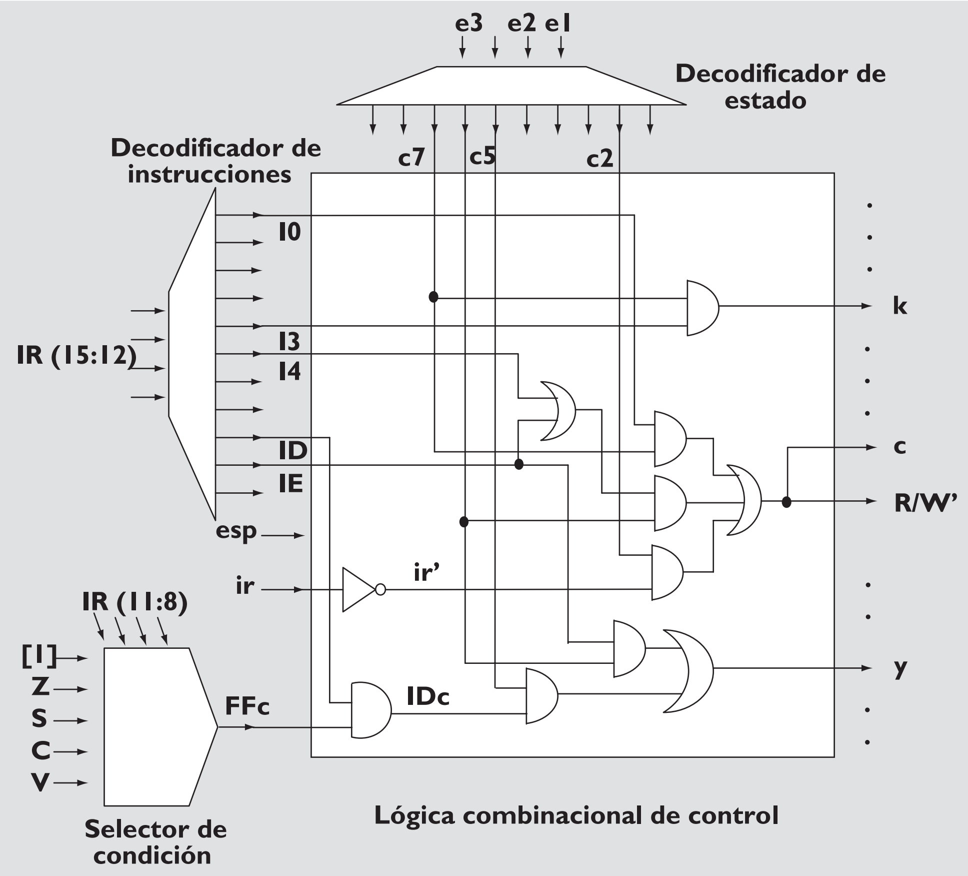


Figura 9.15. *Circuitos que implementa las microórdenes c, k, y, R/W'.*

9.3.2 Unidad de control microprograda de CODE-2

El tipo de unidad de control diseñada para CODE-2 se denomina **unidad de control cableada**, ya que tradicionalmente se diseña a base de circuitos combinacionales y secuenciales interconectados ("cableados"). Una alternativa a las unidades de control cableadas son las unidades de control microprogramadas, ideadas por Wilkes en 1953.

Si se analiza globalmente el funcionamiento de una unidad de control, se observa que lo que hace es generar, sincrónicamente con los pulsos de reloj, vectores de señales de control (microórdenes), que implementan determinadas microoperaciones. En el caso de CODE-2, en cada pulso de reloj se genera un vector de 29 microórdenes *cl, a1, a2, b, ...* $IO/\overline{M}$, $R/\overline{W}$. Los vectores concretos a generar dependen del código de operación de la instrucción en ejecución (*IR(15:12)*), del estado máquina (*c0,...,c9*), y de los valores de los biestables indicadores de condición (*Z, S, C, V*), biestable de espera (*esp*) y biestable de petición de interrupción (*ir*).

Las idea de Wilkes consiste en grabar en una memoria ROM (que se denominará *MC* o **memoria de control**, para distinguirla de la memoria principal) los vectores de microórdenes. De esta forma una unidad de control microprogramada (Figura 9.16) consiste en una memoria de control y en un conjunto de circuitos, denominado **secuenciador de la memoria de control**, que genera las direcciones de las posiciones de las palabras de la memoria de control a leer.

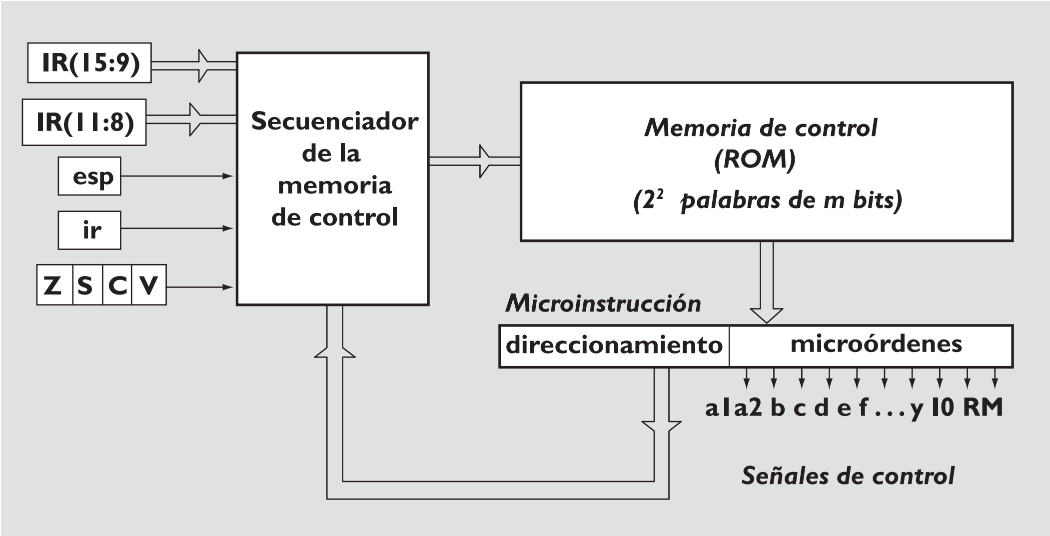


Figura 9.16. *Esquema de una unidad de control microprogramada.*

En realidad, y en el caso del CODE-2, el secuenciador tiene que ir generando secuencias de vectores según el diagrama de flujo de la Figura 9.12; es decir, debe de actuar como un contador con posibilidades de saltos condicionales e incondicionales. Denominamos *DMC* al registro que almacena la dirección de la memoria de control. Si la unidad de control acaba de generar un vector de microórdenes (por ejemplo, el correspondiente a *c0* en la Figura 9.12) el secuenciador debe producir la dirección del siguiente vector, que puede ser el que se encuentra grabado a continuación en la MC (el correspondiente a la microinstrucción $AR \leftarrow alu \leftarrow PC$) en cuyo caso únicamente debe incrementar en uno la dirección de la MC, o, si $ir = 1$ debe producir un salto en la lectura de la MC a la posición donde se encuentra el vector correspondiente a la microinstrucción $IR \leftarrow H'D000$. Como la dirección de salto es función del vector concreto que se acaba de leer, en la MC se graba en una misma palabra, junto al vector de microórdenes, información adicional (que denominaremos *tipo de direccionamiento* o *TD*) indicando cómo el secuenciador debe generar la siguiente dirección, y se reserva además un espacio específico para *direcciones de salto* (campo *DS*), para el caso en que se tenga que producir éste.

Se denomina **microinstrucción** al conjunto de bits correspondientes a las microórdenes que se ejecutan al mismo tiempo junto con los bits que caracterizan o determinan la dirección de la microinstrucción siguiente. En cada palabra de la MC se almacena una microinstrucción. En el caso de CODE-2 cada microinstrucción tiene los cuatro siguientes campos:

TD	o *tipo de direccionamiento*
BE	es un campo para indicar de forma codificada el biestable que impone la condición de salto
DS	indica la dirección de salto, caso de no generarse la nueva dirección por incremento de DMC
microórdenes	contiene 29 bits correspondientes a las 29 señales de control a generar

Una unidad de control microprogramada genera secuencias de microinstrucciones o microprogramas. Un **microprograma** es una secuencia de microinstrucciones que capta o interpreta

una instrucción del lenguaje máquina del computador. Todo lo relativo a los microprogramas suele conocerse como **firmware** (es una frontera entre el hardware y software).

A continuación vamos a describir una posible unidad de control microprogramada para CODE-2. Los tipos de direccionamiento para una unidad de control que implemente las bifurcaciones del organigrama de la Figura 9.12, son los que se indican en la Tabla 9.12. En la primera columna de la tabla se indica una codificación de cada tipo de direccionamiento, que precisamente serán los códigos a incluir en el campo **TD** de cada microinstrucción. La combinación *00* corresponde al caso en que la mucroinstrucción siguiente sea la almacenada en la memoria de control inmediatamente después de la actual; es decir, en este caso el secuenciador tendrá que hacer $DMC \leftarrow DMC + 1$. Obsérvese que la combinación *01* corresponde al momento de la fase de ejecución, en el que hay que seleccionar las microinstrucciones netamente específicas de la instrucción de que se trate; es decir, la nueva dirección de la memoria de control depende (es función) del código de operación, que estará en los 4 bits más significativos del registro instrucción; es decir, en *IR(15:12).*

Tabla 9.12. *Tipos de direccionamiento para obtener la microinstrucción siguiente.*

Campo "tipo de direccionamiento" de la microinstrucción		
TD (tipo de direccionamiento)	**Descripción**	**Explicación**
00	DMC ← DMC + 1	Se incrementa en 1 el registro de dirección de MC
01	DMC ← f[IR(15:12)]	La nueva dirección se obtiene en función del codop
10	DMC ← DS	Salto incondicional
11	Si CS = 1; DMC ← DS Si CS = 0; DMC ←DMC + 1	Salto condicional

La combinación *TD = 10* la hemos reservado para el caso en que haya que efectuar un salto incondicional a otra microinstrucción; tal es el caso de la última microinstrucción de ejecución de una instrucción, ya que en esta situación hay que saltar al inicio del diagrama de flujo (estado *c0* de la Figura 9.12). El código *TD = 11* corresponde a los saltos condicionales. Analizando con detalle la Figura 9.12 se puede concluir que los saltos condicionales, dependiendo de la microinstrucción de que se trate, se producen en función de las variables *esp, ir* y *FFc*. Recuérdese que *FFc* es sencillamente el valor del biestable indicador que impone la condición del salto en las instrucciones máquina de bifurcación, y que puede obtenerse con un selector de datos controlado por los bits del campo de condición de la instrucción, *IR[11:8].* Como en cada microinstrucción que implique la realización de un salto condicional tenemos que especificar la variable (biestable) que marca la condición (en el campo *BE*), codificamos las variables según se indica en la Tabla 9.13. El campo *BE*, tendrá por tanto 2 bits, y la condición seleccionada la denominaremos *CS.*

Si la memoria de control contuviese *m* microinstrucciones, el campo de dirección de salto *DS*, tendría *n* bits, siendo $m = 2^n$. Como veremos más adelante, nuestra memoria de control ocupa m = 256 palabras (no todas ellas ocupadas); es decir, $n = 8$ bits. Teniendo en cuenta el tamaño de cada campo, la longitud de cada microinstrucción (o tamaño de palabra de la memoria de control) es: $2 + 2 + 8 + 29 = 41$ bits.

Tabla 9.13. *Codificación de los biestables de control de saltos.*

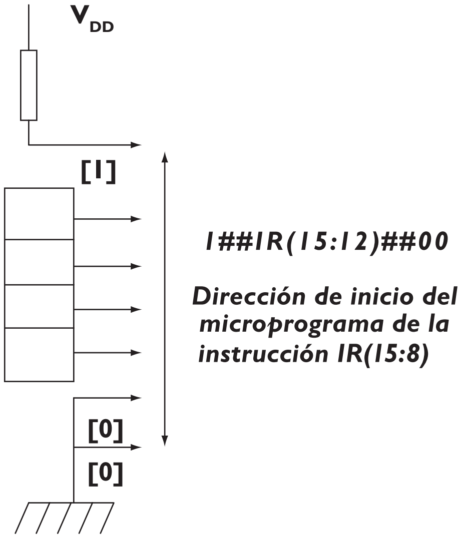
Campo "condición de salto" de la microinstrucción		
BE	**Biestable que impone la condición**	**Explicación**
00	Esp	Biestable de espera
01	ir	Biestable de petición de interrupción
10	FFc	Biestable indicador de la ALU (Z, S, C o V)
11		

En nuestro diseño queda por establecer cómo se obtiene la dirección de la memoria de control (*DMC*) a partir de la que se encuentran las microinstrucciones específicas para ejecutar cada instrucción máquina, que denominaremos **microprograma de ejecución**. En definitiva se trata de encontrar el valor de DMC en función del código de operación: $IR \leftarrow f(IR[15:8])$ (vése segunda fila de la Tabla 9.12). Éste es un problema complicado ya que debe resolverse lo más veloz posible. Piénsese que la unidad de control microprogramada tiene que ser extremadamente rápida ya que la ejecución de cada instrucción del repertorio máquina implica la ejecución de varias microinstrucciones, y éstas por tanto van a condicionar la velocidad máxima posible del procesador. Una forma de implementar la transformación del código de operación (codop) en la dirección de inicio de microprograma de ejecución, es grabar en una ROM una tabla de consulta. Cada codop corresponde a una dirección de una posición de la tabla que contendrá directamente la dirección de la memoria de control correspondiente. Con este procedimiento, para obtener el tiempo de producción de microinstrucciones habría que sumar al tiempo de acceso de la memoria de control el tiempo de lectura de la tabla de direcciones, salvo que se prevea algún mecanismo de paralelización.

En el caso de CODE-2 hemos optado por un sistema más ingenioso, ya que es más rápido y sencillo de implementar. En la Figura 9.12 observamos que la ejecución de cada instrucción (una vez captados los operandos) supone en 15 de los 16 casos la ejecución de 4 o menos microinstrucciones (corresponden a los ciclos *c5* a *c8*, de la figura citada); en otras palabras podemos reservar para cada microprograma de ejecución cuatro microinstrucciones (en el caso del microprograma de la instrucción *ID*, llamada a subrutina, generaríamos un salto incondicional a otra zona de la memoria de control). Podemos por tanto situar en direcciones múltiplos de 4 consecutivas los *comienzos* de los microprogramas de ejecución. De esta manera podríamos asignar a la instrucción *I0* la dirección *0000**00***, a la *I1* la dirección *0001**00***, a la *I2* la dirección *0010**00***, etc. Obsérvese que estamos formando las direcciones de la memoria de control con el código de operación (*0000* de la instrucción *I0*, *0001* de la instrucción *I1*, etc., véase Tabla 9.8) añadiéndole dos ceros (***00***) al final. Por otra parte nos interesa reservar las direcciones bajas de memoria para almacenar las microinstrucciones de captación de instrucción y de operandos (ciclos *c0* a *c4*, en la Tabla 9.12), por lo que muy bien podríamos anteponer un *1* a las posibles direcciones de los microprogramas de ejecución. En definitiva las direcciones buscadas serían: instrucción *I1*, ***1**0000**00***; instrucción *I2*, ***1**0001**00**;* ...; instrucción *IF*, ***1**1111**00*** (Figura 9.17a). Los microprogramas, de esta forma, ocuparán de la posición 64 a 127 de la memoria de control. La implementación de esta técnica es trivial: consiste en añadir a la información de salida del registro instrucción (donde se encuentra codop) una conexión a la alimentación (V_{DD}), para obtener un 1 lógico que actuaría como bit más significativo, y dos conexiones a tierra, para obtener dos ceros lógicos, que actuarían como bit menos significativos (Figura 9.17b).

Dirección de la memoria de control		*Contenido de la memoria de control*
64	**1 0000 00**	Microprograma instrucción I0
65	**1 0000 01**	
66	**1 0000 10**	
67	**1 0000 11**	
68	**1 0000 00**	Microprograma instrucción I1
69	**1 0000 01**	
70	**1 0000 10**	
71	**1 0000 11**	
72	**1 0000 00**	Microprograma instrucción I2
73	**1 0000 01**	
74	**1 0000 10**	
75	**1 0000 11**	
⋮	⋮	
124	**1 0000 00**	Microprograma instrucción IF
125	**1 0000 01**	
126	**1 0000 10**	
127	**1 0000 11**	

(a)



(b)

Figura 9.17. *Obtención de la dirección de la memoria de control correspondiente a cada instrucción: (a) asignación de direcciones; (b) implementación.*

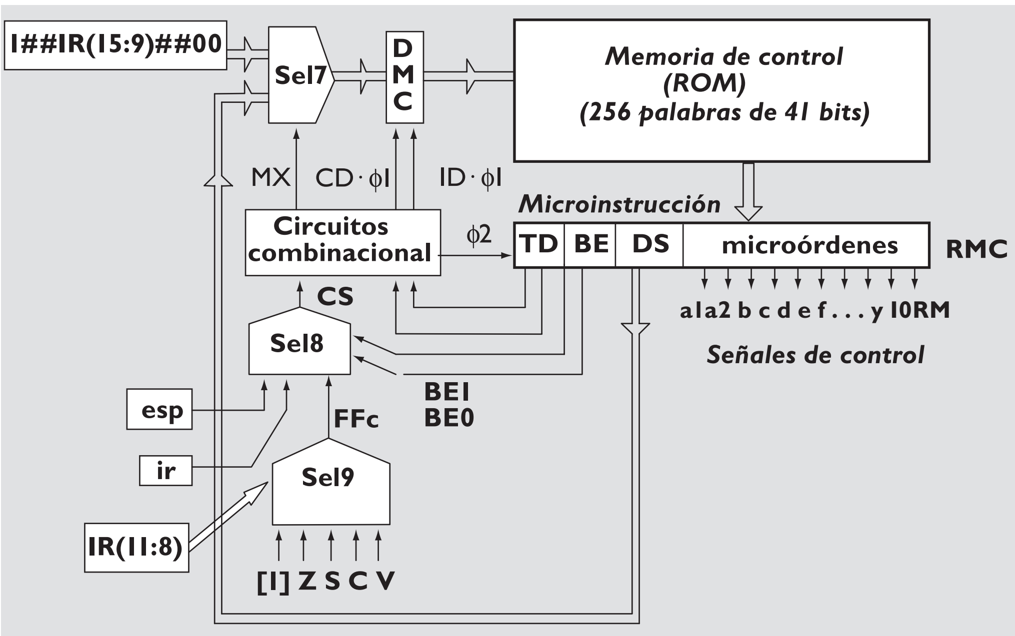


Figura 9.18. *Unidad de control microprogramada de CODE-2.*

La estructura de la unidad de control diseñada es la que se muestra en la Figura 9.18. El contador-registro de dirección de memoria *(DMC)* en algunas situaciones debe incrementarse una unidad, acción que se realiza con la señal *ID* o, en otras, cargarse en paralelo, bien con la dirección procedente de *IR[15:8]*, o con la dirección *DS* contenida en el registro de la microinstrucción, *RMC*. Esto nos obliga a incluir un selector (*SEL7*, en la figura) a la entrada de *DMC*, que se controlará con la señal *MX*. El contador *DMC* se carga en paralelo con la señal *CD*.

Hay que incluir unos circuitos combinacionales, que generen las señales *MX, CD* e *ID*. Además esas señales van a depender del valor de los biestables de condición, con lo que seleccionamos (con el selector Sel8) la salida del biestable que impone la condición de salto. La salida seleccionada la denominamos *CS*, y, obviamente, los controles de selección los podemos obtener directamente del campo *BE,* tal como se muestra en la Figura 9.18. Como se hizo en la unidad de control cableada, utilizamos otro selector (*Sel9*) para obtener (en función del campo de condición de las instrucciones de bifurcación; es decir, de los bits *IR[11:8]*) el valor, *FFc*, del biestable indicador de la *ALU* que impone la condición de salto.

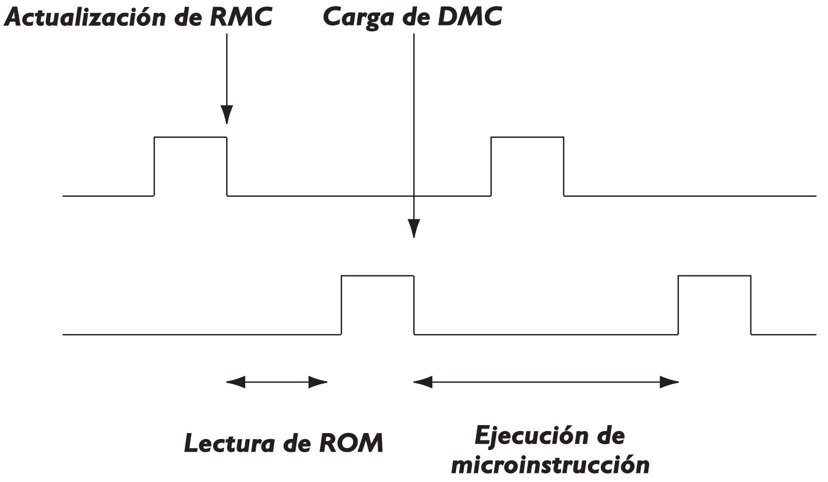


Figura 9.19. *Temporizador para la unidad de control microprogramada de CODE-2.*

El funcionamiento de la unidad de control microprogramada hay que temporizarlo adecuadamente. Una forma fácil de hacerlo es partir de dos señales de reloj desfasadas, como se indica en la Figura 9.19. Con sincronismo Φ_1 se actualiza el registro de dirección de la memoria de control (*DMC*), con Φ_2 se carga el registro de microinstrucción (*RMC)* con la palabra leída de la memoria. El desfase mínimo entre Φ_1 y Φ_2 vendrá impuesto por el tiempo de acceso de la memoria de control, y el desfase entre Φ_2 y Φ_1 del tiempo que tarde en ejecutarse la microinstrucción. De acuerdo con lo anterior, las señales de carga y avance de DMC se han sincronizado con Φ_1 y la carga del registro de microinstrucción con Φ_2 (Figura 9.18).

Las funciones que sintetizan las señales de control son:

Selección de direcciones

MX debe ser 1 para el tipo de direccionamiento en que DMC ← f(RI); es decir (Tabla 9.12):

$$MX = \overline{TD(1)} \cdot TD(0)$$

Incremento de DMC

El registro de dirección de memoria debe incementarse; es decir, la señal de incremento de dirección *(ID)* debe ser 1, cuando (Tabla 9.12): *TD = 00,* y cuando TD = 11 siempre que no se produzcan saltos *(CS = 0);* es decir:

$$ID = \overline{TD(1)} \cdot \overline{TD(0)} + TD(0) \cdot TD(1) \cdot \overline{CS}$$

Carga en paralelo de DMC

La señal de carga de dirección *(CD)* debe ser 1 (Tabla 9.12) en los casos donde no haya que incrementar DMC; es decir:

$$CD = \overline{ID}$$

En la Figura 9.20 se muestra un esquema de los circuitos que sintetizan las funciones anteriores.

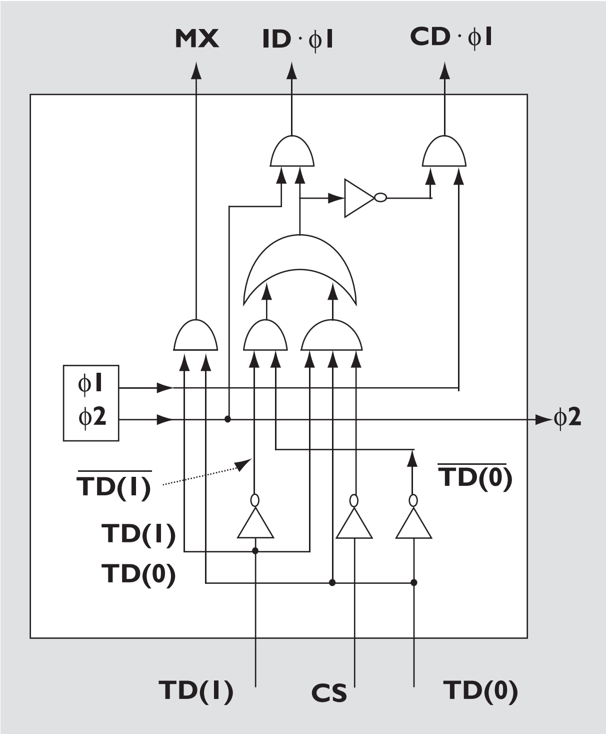


Figura 9.20. *Esquema de los circuitos combinacionales del secuenciador de la memoria de control.*

Una vez diseñado el secuenciador de direcciones, nos queda realizar los microprogramas correspondientes. Éstos se confeccionan fácilmente con ayuda del organigrama de la Figura 9.21, que hemos obtenido a partir del de la Figura 9.12, adaptándolo a la filosofía de funcionamiento de la unidad de control microprogramada que hemos diseñado. La Tabla 9.14 muestra un fragmento de los microprogramas a memorizar en la memoria ROM de control. Las señales de control *a1, a2, b, c, cl, ...*$IO/\overline{M}$, $R/\overline{W}$ se han determinado de acuerdo con la microoperación a realizar (segunda columna de la Tabla 9.9).

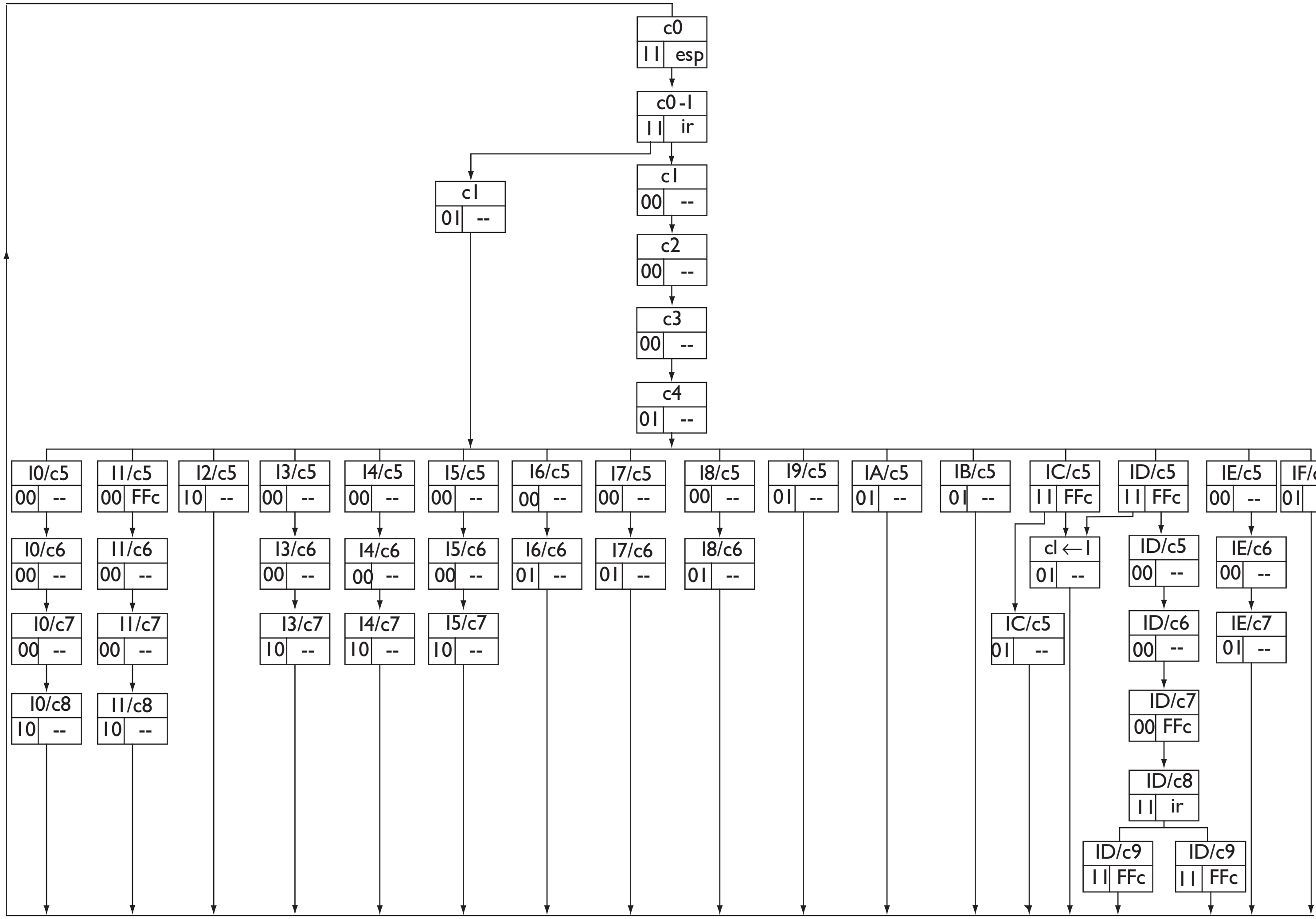


Figura 9.21. *Organigrama de los microprogramas de la unidad de control de CODE-2.*

Tabla 9.14. *Fragmento de los microprogramas de la memoria de control de CODE-2 (puede verse el contenido completo en la web del libro.*

Dirección		Instrucción	Microoperación	Microinstrucción																						
				Direccionamiento			Microórdenes																			
				TD	BE	DS	a1	a2	b	c	cl	d	e	esp_st	f	g	h	i	ir1	ir2	ir_cl	ir_st	k	...	IO/M’	R/W’
0	0 0000 00	captación	¿esp=1?	11	00	000 0000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0	1
1	0 0000 10		¿ir=1?	11	01	000 0110	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0	1
2	0 0000 01		AR←PC	00	--	--------	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	…	0	1
3	0 0000 11		DR←M; RT←PC	00	--	--------	0	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	...	0	1
4	0 0001 00		PC←RT+1	00	--	--------	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	...	0	1
5	0 0001 01		IR←DR	01	--	--------	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	…	0	1
6	0 0001 10		IR←H’D000	01	--	--------	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	…	0	1
..	....	………	……................…………………	…..	…	……	………………………………………………																			
64	1 0000 00	I0	RA←H’D;RT←RF	00	--	--------	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	...	0	1
65	1 0000 01		AR←00#IR(7:0)+RT	00	--	--------	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	…	0	1
66	1 0000 10		DR←M	00	--	--------	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	…	0	1
67	1 0000 11		WA←rx,RF←DR,cl←1	10	--	000 0000	0	1	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	…	0	1
68	1 0001 00	I1	RA←H’D;RT←RF	00	--	--------	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	...	0	1
69	1 0001 01		AR←00#IR(7:0)+RT	00	--	--------	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	...	0	1
70	1 0001 10		RA←rx,DR←RF	00	--	--------	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	...	0	1
71	1 0001 11		M←DR, cl←1	10	--	000 0000	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	...	0	0
72	1 0010 00	I2	WA←rx, RF←00#IR(7:0), cl←1	10	--	000 0000	0	1	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	...	0	1
73	1 0010 01																									
74	1 0010 10																									
75	1 0010 11																									
76	1 0011 00	I3	RA←rs;WA←rx,RF←00#RF(7:0)	00	--	--------	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0	1
77	1 0011 01		RA←rx,RT←RF	00	--	--------	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	...	0	1
78	1 0011 10		WA←rx, RF←IR(7:0)#00+RT, cl←0	10	--	000 0000	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	...	0	1
79	1 0011 11																									
..	....	………	……................…………………	…..	…	……	………………………………………………																			
124	1 1111 00	IF	esp←1; cl←1	10	--	000 0000	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	...	0	1
125	1 1111 01																									
126	1 1111 10																									
127	1 1111 11																									

En los computadores grandes las memorias de control pueden contener varios miles de palabras, con microinstrucciones del orden de 100 bits. En CODE-2 los bits del campo de señales de control actúan directamente sobre los elementos que controlan; este tipo de implementación se denomina ***microprogramación horizontal***. A veces, para conseguir que la longitud de las microinstrucciones sea menor, las señales se agrupan y codifican en campos específicos, en este caso las señales que actúan directamente sobre los distintos elementos se obtienen a partir de la microinstrucción con ayuda de decodificadores, denominándose esta implementación ***microprogramación vertical***.

Las unidades de control microprogramadas son más fáciles de diseñar y de actualizar que las cableadas; pero, por contra, en general son más lentas que estas últimas.

9.4 Memoria

Según se indicó en la Sección 1.2. la memoria es una unidad donde se almacena la información que necesita el computador; es decir, tanto las instrucciones que forman los programas como los datos. Por otra parte en las Secciones 5.4.8 y 6.6 se analizó la memoria a nivel de lógica digital.

En esta sección nos referiremos a la memoria principal, concretamente efectuaremos el diseño de la memoria de CODE-2. Recuérdese (Sección 7.1) que para seleccionar una palabra en la memoria principal debe especificarse su dirección, se dice que son **memorias accesibles por dirección**. A veces, también se utiliza **memorias accesibles por contenido** o **memorias asociativas**, donde en lugar de una dirección, se da una parte del contenido de la posición (denominada **clave**) y la memoria proporciona la totalidad del contenido de las palabras que contienen esa clave. Aquí nos referiremos a las memorias accesibles por dirección.

Como se analizó en la Sección 1.2., la memoria principal de un computador suele estar constituida por dos tipos de memoria: ROM y RAM. En los Capítulos 5 y 6 presentamos algunos conceptos sobre estos dos tipos de memoria (Secciones 5.4.8 y 6.6).

En la Figura 9.22a se muestra un esquema de cómo se desea organizar la memoria de CODE-2, que es de 64 K palabras de 16 bits. De esta capacidad las 62 K palabras primeras (direcciones *H'0000* a H'F7FF) deben ser de memoria RAM; y el resto (posiciones de la *H'F800* a la *H'FFFF*) de memoria ROM.

Por lo general el problema del ingeniero consiste en configurar una memoria con una determinada capacidad y longitud de palabra, utilizando circuitos integrados comercialmente disponibles. Examinando las disponibilidades de circuitos de memoria encontramos los chips IS61C6416 y CY7C1021, que son memorias RAM estáticas organizadas en la forma 64K x 16 bits, y que por tanto se adaptan plenamente a los requisitos de CODE-2. Realmente sólo necesitamos 62 K palabras, pero comercialmente no existe un chip con esa concreta capacidad.

Para memoria ROM podemos utilizar el circuito integrado 27C64, organizado en 8 K palabras de 8 bits. La longitud de palabra de 16 bits la conseguimos utilizando dos chips en paralelo.

En la Figura 9.22b se muestra las zonas de memoria que se aprovecharía de cada circuito integrado.

Para el control de la memoria debemos utilizar las señales $IO/\overline{M}$, y $R/\overline{W}$ del bus de control, que por simplificar la notación vamos a denominarlas *IO* y *R*. Recuérdese que cuando el procesador tiene que leer de memoria genera las señales: $IO = 0$ y $R = 1$; y cuando tiene que escribir

en memoria: *IO = 0* y *R = 0*. El diseño de la memoria se centra ahora en adaptar estas señales a las especificaciones de los circuitos concretos utilizados.

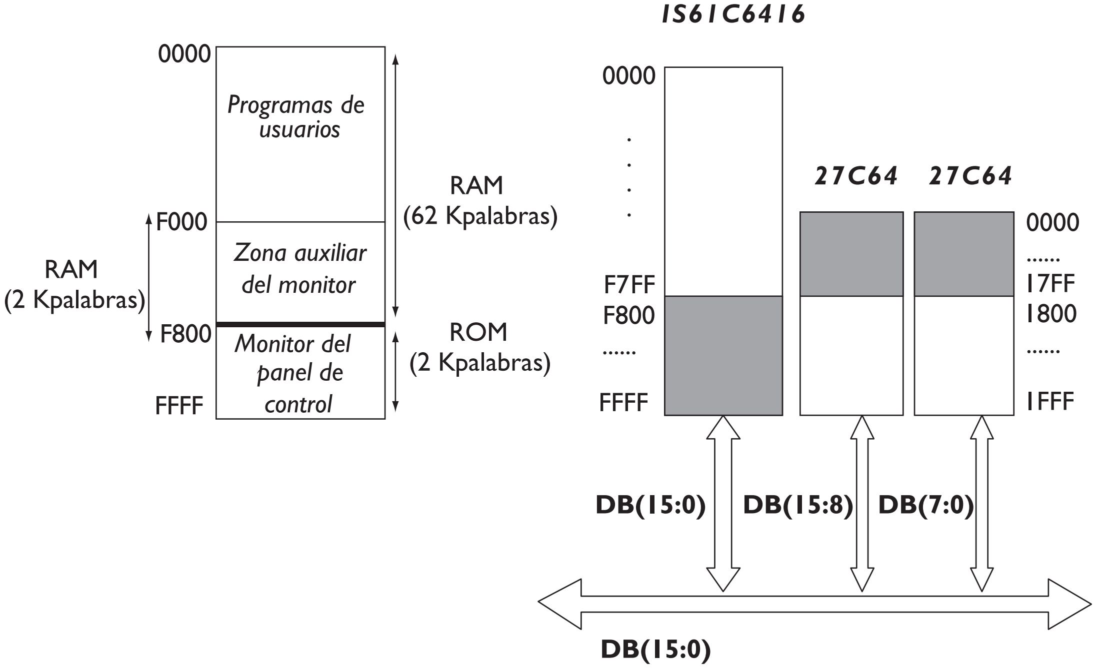


Figura 9.22. *(a) Organización de la memoria de CODE-2; (b) Organización utilizando circuitos comerciales.*

Concretamente, en las hojas de características del chip IS61C6416 [webMEM] se indica lo siguiente:

1) Existe un único bus para entrada y para salida de datos (es un bus bidireccional), de 16 bits: I/O(15:0). Se puede seleccionar la entrada o salida del byte superior o inferior de la palabra; esto se hace por medio de las entradas de control $\overline{LB}$ (byte inferior) y $\overline{UB}$ (byte superior).
2) Es obvio que el bus de direcciones es de 16 bits: A(15:0).
3) Además de las dos señales de control indicadas, dispone de las siguientes:
   - $\overline{CE}$, habilitar el chip; si esta señal es 1 la memoria no actúa, ni grabando ni leyendo información (las salidas de la memoria están en alta impedancia y la memoria no actúa sobre el sub-bus de datos del sistema).
   - $\overline{OE}$, habilitar salida; debe hacerse 0 cuando se desee leer.
   - $\overline{WE}$, habilitar entrada; debe hacerse 0 cuando se desee escribir.
3) La alimentación que requiere es de 5 voltios.
4) Existen distintas versiones, con tiempos de acceso de 10, 12, 15 y 20 nanosegundos.

De acuerdo con las especificaciones para las señales $\overline{LB}$, $\overline{UB}$, $\overline{CE}$, $\overline{OE}$ y $\overline{WE}$, y el comportamiento de las señales IO y R generadas por el procesador, podemos obtener con ayuda de la Tabla 9.15 aquellas señales en función de éstas.

Tabla 9.15. *Señales de control del chip de memoria en función de las señales de control generadas por el procesador.*

$IO/\overline{M}$	$R/\overline{W}$	Operación	$\overline{CE}$	$\overline{OE}$	$\overline{WE}$	$\overline{LB}$	$\overline{UB}$
0	0	Escribir en memoria (16 bits)	0	1	0	0	0
0	1	Leer de memoria (16 bits)	0	0	1	0	0
1	0	Escribir en puerto de salida	1	x	x	x	x
1	1	Leer de puerto de salida	1	x	x	x	x

De la tabla, y siguiendo el procedimiento descrito en la Sección 6.3, se deducen fácilmente las señales de control de la memoria son:

$$\overline{CE} = IO/\overline{M}$$

$$\overline{OE} = \overline{R/\overline{W}}$$

$$\overline{WE} = R/\overline{W}$$

$$\overline{LB} = IO/\overline{M}$$

$$\overline{OB} = IO/\overline{M}$$

Es decir, directamente con las señales del sub-bus de control se puede gestionar adecuadamente la memoria, necesitándose tan sólo un inversor para la señal $\overline{OE}$.

Ahora bien, según indicamos anteriormente (Figura 9.22b) debemos seleccionar la memoria RAM cuando en el bus de direcciones, *AB(15:0),* se encuentre una dirección comprendida entre *H'0000* y *H'F7FF*, y la memoria ROM cuando la dirección este entre H´F800 y H'FFFF. En otras palabras, cuando la función *AB(15) · AB(14) · AB(13) · AB(12) · AB(11)* sea igual a 1 debe seleccionarse la ROM; y en caso contrario, es decir, cuando *AB(15) · AB(14) · AB(13) · AB(12) · AB(11) = 0*, se debe seleccionar la RAM, con lo que la señal de selección de la RAM debe ser:

$$\overline{CE}_{RAM} = IO/\overline{M} \cdot \overline{AB(15) \cdot AB(14) \cdot AB(13) \cdot AB(12) \cdot AB(11)}$$

El circuito de memoria ROM elegido tiene sólo tres señales de control, $\overline{CE}$, $\overline{OE}$ y $\overline{PGM}$, coincidiendo el significado de las dos primeras con las del mismo nombre de la RAM, y la última, que debe ser 0 para programar la memoria (es una memoria programable: PROM). En consecuencia las funciones que sintetizan las señales de control de la ROM, son:

$$\overline{CE}_{ROM} = IO/\overline{M} \cdot \overline{AB(15) \cdot AB(14) \cdot AB(13) \cdot AB(12) \cdot AB(11)}$$

$$\overline{OE} = \overline{R/\overline{W}}$$

$$\overline{PGM} = 1$$

En la Figura 9.23 se muestra un esquema del diseño realizado para la memoria.

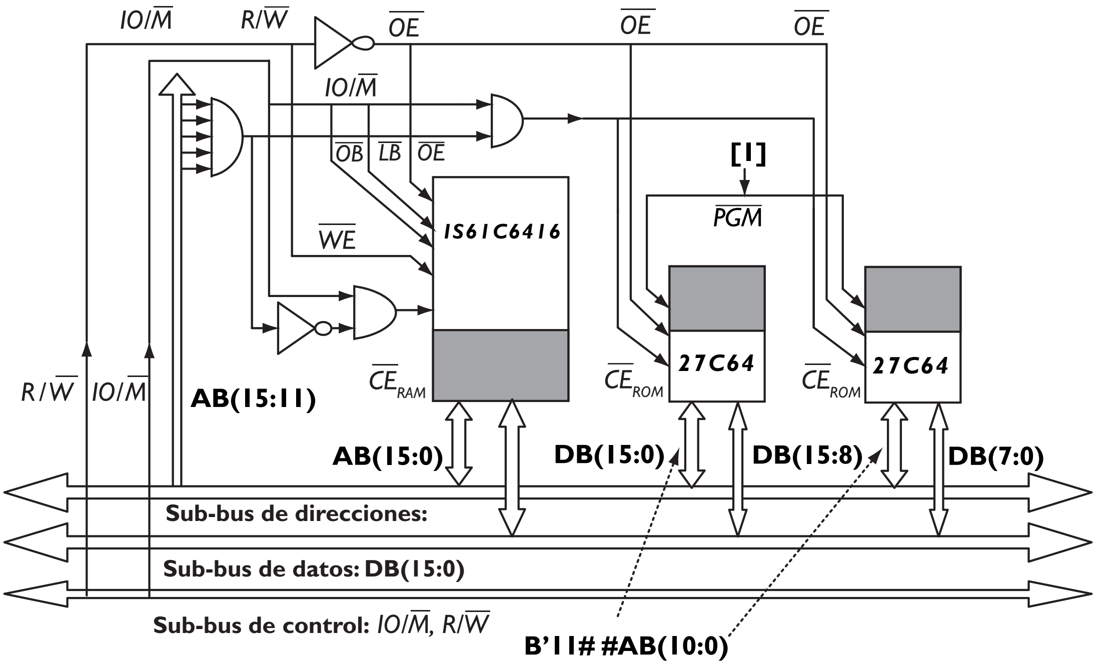


Figura 9.23. *Circuitos que generan las señales de control de las memorias RAM y ROM.*

9.5 Organización de entradas/salidas

Hasta ahora hemos hecho hincapié en el estudio del procesador y la memoria principal de un computador. En esta sección, dedicada a **organización de entradas/salidas**, se van a analizar los problemas y soluciones que presenta la conexión del procesador de un computador a una gran variedad de dispositivos periféricos. En los Capítulos 10 y 11 se estudiarán los periféricos propiamente dichos.

El problema de conexión de periféricos a un procesador se debe principalmente a que las características de los dispositivos de E/S suelen diferir considerablemente de las del procesador, entre otras:

- La velocidad de transmisión de los periféricos es notablemente menor que la velocidad con que opera el procesador, y además es muy variable (desde unos 5 Bytes/s a más de 1 MByte/s).
- La longitud de las palabras de datos suelen variar, y
- Los códigos y niveles eléctricos para representar los datos suelen diferir.

Para comprender adecuadamente las funciones de los elementos electrónicos de un periférico conviene distinguir dos niveles en la realización de una Entrada/Salida:

- **Transferencias elementales de información.** Sirven para la recepción o envío de una información individual (byte o palabra). Estas informaciones transmitidas por el sub-bus de datos pueden ser datos propiamente dichos o información de control para el periférico o el procesador, o información sobre el estado del propio periférico. Las transferencias

elementales de información se realizan físicamente a través de **puertos de entrada** o **salida**, que son registros que se conectan directamente a uno de los buses del computador. Cada puerto tiene asociado una dirección o código, de forma que el lenguaje máquina (el procesador) "ve" a un periférico como si fuese un puerto o conjunto de puertos.

- **Operaciones de entrada/salida.** Una operación de entrada consiste en la transferencia de un conjunto de datos (usualmente denominado **bloque** o **registro físico**) que constituye una línea del monitor de pantalla, o un sector de un disco, o un bloque de una cinta magnética, por ejemplo. Las operaciones de E/S se componen, por lo tanto, de transferencias elementales que son monitorizadas por el procesador o por circuitos incluidos en el **controlador del periférico**.

9.5.1 Controladores de E/S

Para compatibilizar las características de los dispositivos de E/S con las de un procesador e intercambiar información entre ambos, se usan **controladores de periféricos** (circuitos de interfaz). Cada periférico necesita disponer de su propio controlador que puede ser más o menos complejo. En la Sección 5.5 (Figuras 5.18 a 5.20) se mostraron distintas estructuras de computador que incluían distintos tipos de periféricos con controladores.

Un **controlador** está constituido por un conjunto de circuitos de *adaptación* del formato de señales y de los parámetros de temporización entre el procesador y los dispositivos de E/S. El controlador también se encarga de llevar a cabo las *transferencias de datos* entre el procesador y el periférico; para ello se conecta entre el periférico que controla y un bus del computador (bus de entrada/salida, por ejemplo), y recibe señales de control (a través del sub-bus de control), tales como $R/\overline{W}$ e $IO/\overline{M}$, y genera señales de control para el periférico y señales de estado (ocupado/listo, error de paridad, etc) para el procesador.

Desde el procesador se actúa sobre los periféricos a través de las instrucciones de los programas. Para poder utilizar eficazmente un periférico, su sistema operativo debe contener una rutina especial denominada **gestor del periférico** (véase Sección 13), que controla las transferencias de información. El conjunto de gestores de periféricos suele denominarse **Sistema de Control de Entrada-Salida** o ***IOCS*** (*Input-Output Control System*). Sin estas rutinas del sistema operativo sería imposible o extremadamente complejo utilizar un periférico desde un lenguaje de alto nivel. En esta sección sólo se hace hincapié en los aspectos físicos relacionados con los periféricos, dejando los aspectos relacionados con su gestión software para la Sección 13.

Los controladores de periféricos se interconectan al bus (local, del sistema o de E/S, véase Sección 12.5) a través de puertos de E/S. En la Figura 9.24 se muestra el esquema de un registro utilizable como puerto. Obsérvese que contiene una entrada para una señal de carga paralela (*S*, o *Strobe*), y la salida es tri-estado, de forma que sus salidas permanecen en estado de alta impedancia (desconectadas eléctricamente del bus) cuando la señal de control $T = 0$, y en estado de baja impedancia (0 ó 1, conectadas al bus) cuando $T = 1$. De esta forma se pueden conectar físicamente las salidas de varios puertos al mismo bus, y controlar su conexión eléctrica con la señal *T*. Existen circuitos integrados con los que se pueden formar conjuntos de registros del tipo mencionado; así con 8 circuitos *9334* se pueden formar 4 registros direccionables individualmente de 8 bits cada uno. Piénsese que en caso de no utilizar registros tri-estado para conectar los puertos de entrada al bus de datos habría que utilizar multiplexores, lo que aumentaría considerablemente la circuitería, el número de hilos de interconexión y la expansión para incluir nuevos puertos sería difícil.

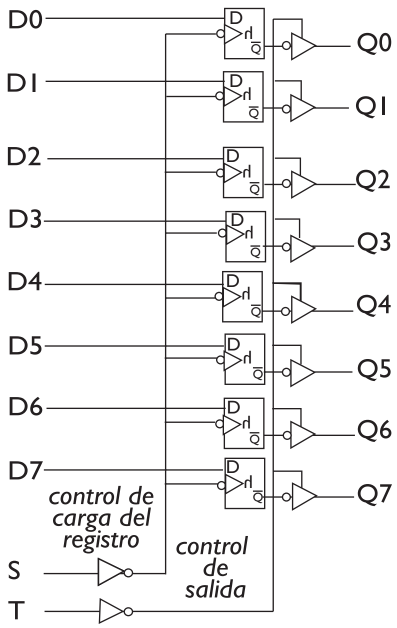


Figura 9.24. *Esquema de un posible puerto de entrada/salida.*

Los controladores suelen contener seis elementos: selector de direcciones, puerto de datos, puerto de estado, puerto de control, lógica de interfaz con el periférico, y circuitos que generan las secuencias de control (**secuenciador del controlador**). En la Figura 9.25 se muestra un diagrama funcional de un posible controlador. Hay que hacer notar que los controladores no siempre disponen de todos los elementos que se indican en la figura y, además, pueden contener otros adicionales no incluidos en ella. Así, no es raro que haya un único puerto para la información de estado y de control; y si, por ejemplo, el puerto es de salida no exista el puerto de dato de entrada. Los controladores cubren básicamente estos objetivos:

1) *Selección o direccionamiento del periférico*. El procesador sitúa en el sub-bus de direcciones el código o dirección del puerto que debe intervenir en el tráfico de datos. En una transmisión concreta, sólo uno de los puertos, P_i, debe estar conectado *eléctricamente* al bus de datos, para transmitir a través de él (haciendo en él $T_i = 1$, en puertos de entrada, o $S_i = 1$, en puertos de salida). El **selector de direcciones** se encarga de identificar cuándo la dirección del sub-bus de direcciones corresponde a su código (dirección), para dar paso al intercambio de información con el periférico. De esta forma se evitan posibles conflictos de acceso al bus.
2) *Almacenamiento temporal.* La comunicación física entre el periférico y el procesador se hace a través del controlador, por lo que éste contiene un **puerto de datos** para almacenar temporalmente los datos a transferir.
3) *Sincronización.* La velocidad operativa de las unidades centrales es mucho mayor que la de los periféricos. El controlador regula el tráfico de información para que no se den problemas de sincronización o de pérdidas de información. El controlador suele actuar con unas señales de control y estado que intercambia con el procesador, indicando situaciones tales como que está preparado o listo (*ready*) para recibir o transmitir, que ha reconocido (*acknowledge*) la llegada de unos datos, que desea ser atendido por el

procesador (señal de petición de interrupción) para transmitir a ésta unos datos, etc. En definitiva, el secuenciador del controlador en función del contenido del puerto de estado y control debe generar señales de estado y control para iniciar o terminar la transferencia (*listo, petición, reconocimiento,...)*.

4) *Control del periférico.* El procesador debe ser capaz de interrogar al controlador, para conocer su estado y el del periférico, leyendo el contenido del **puerto de estado**, o de enviar órdenes a éste, escribiéndolas en el **puerto de control**. La operación de lectura suele realizarse siempre inmediatamente antes de transferir o captar datos del periférico. Usualmente se puede acceder individualmente a los bits del puerto de estado y control, pudiendo unos bits ser activados por el procesador y otros por el propio controlador. Por ejemplo, un bit puede indicar si el dispositivo se encuentra conectado/desconectado a la red de suministro eléctrico; obviamente este bit será activado por el periférico.
5. *Interfaz con el periférico.* Consiste en la adaptación o conversión de las características eléctricas y lógicas de las señales empleadas por el dispositivo de E/S y por el bus:
   - Niveles de tensión, impedancias, etc.
   - Lógica (positiva o negativa).
   - Conversión paralelo/serie y serie/paralelo, etc.
6) *Detección de errores*, mediante la inclusión/detección de paridad, transformación según códigos polinómicos, etc. En caso de detectarse un error de este tipo, el secuenciador cambiaría el bit del puerto de estado indicador de error de transmisión, y se procedería a una repetición de la transferencia de información.
7) *Gestión de transmisión de bloques de información.* En los periféricos que intercambian la información en forma de registros físicos (y no con palabras de datos aisladas), el controlador puede disponer de un contador que controle el número de palabras recibidas/transmitidas.

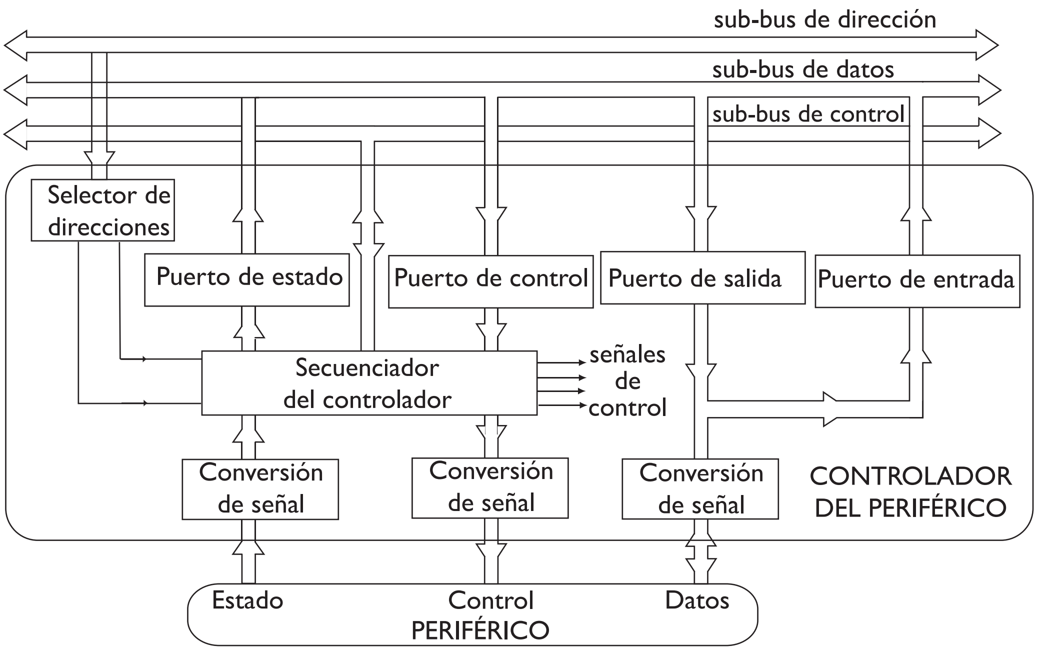


Figura 9.25. *Esquema de un controlador de E/S genérico.*

Los controladores pueden estar físicamente ubicados en el chasis del computador central (controlador gráfico para monitor de video, por ejemplo) o en el dispositivo propiamente dicho (controlador IDE de disco duro), o distribuidos entre ambos.

Las operaciones de entrada/salida son de una gran complejidad. En los primeros computadores, el procesador controlaba minuciosamente todas las operaciones elementales que efectuaba el periférico. Poco a poco estas funciones, para descargar de trabajo al procesador, se han ido descentralizando y pasando a los controladores, resultando éstos cada vez más complejos. Un controlador que realiza operaciones muy complejas se denomina **controlador inteligente**; este sería el caso de los **procesadores gráficos**, que pueden interpretar un lenguaje de alto nivel para realizar dibujos y controlan pantallas gráficas (véase Capítulo 8). La mayor parte de los controladores pueden considerarse computadores de uso específico, ya que contienen un microprocesador, ROM con programas de gestión del periférico, una **memoria intermedia** (*buffer*), o **memoria caché** para datos (256 KB y superiores en algunos controladores de disco), etc. El procesador, a través de los puertos de salida correspondientes, da las órdenes oportunas al controlador para realizar un conjunto de operaciones de entrada o salida.

En los computadores grandes es normal que se incluyan **procesadores de periféricos** (*Input/Output Processor* o **IOP**) o **canales de datos**. Estos son computadores dedicados a funciones de entrada/salida, que suelen contener un microprocesador con instrucciones especializadas y una memoria local. Cada procesador de periféricos puede atender a varios periféricos simultáneamente, encargándose de operaciones de transferencias específicas de E/S tal como intercambio de datos entre dos periféricos conectados a él (copia de un fichero de un disco a otro, por ejemplo). Como en el caso de los controladores inteligentes, siempre es el procesador el que supervisa el funcionamiento del IOP; aquí el procesador, una vez que ha cargado en los puertos, o en una zona de memoria principal accesible por ambos, los parámetros correspondientes, lanza a ejecutar los programas residentes en los procesadores de periféricos (bien fijos en ROM o almacenados previamente por el sistema operativo). Cuando el IOP finaliza la ejecución del programa de E/S indica este hecho al procesador por medio de una interrupción.

9.5.2 Direccionamiento de puertos de E/S

Con referencia a las operaciones de entrada/salida, los repertorios de instrucciones de los computadores suelen optar por una de las dos siguientes posibilidades:

a) *Disponer de instrucciones de E/S específicas.* Así el Pentium dispone, entre otras, de las siguientes instrucciones de E/S:

IN AX, puerto — *AX* ← *puerto* (lleva el contenido del puerto especificado en la propia instrucción al acumulador).

OUT puerto, AX — *puerto* ← *AX* (lleva el contenido del acumulador al puerto especificado en la propia instrucción).

En la Figura 9.26a se muestra un ejemplo de cómo quedaría distribuido el mapa de memoria para un procesador con direcciones de memoria de 16 bits, y direcciones para puertos de salida de 8 bits (caso, por ejemplo, del microprocesador *8085*).

b) *No disponer de instrucciones específicas de E/S.* En este caso los puertos de E/S se codifican utilizando posiciones de memoria, diciéndose entonces que son ***instrucciones de E/S en el mapa de memoria*** (Figura 9.26b). En efecto, en este caso se selecciona un conjunto de direcciones de memoria para codificar las direcciones de los puertos. El selector de direcciones de cada puerto o del controlador (Figura 9.25) se encarga de comprobar si la dirección (en principio de memoria) depositada en el sub-bus de direcciones

corresponde a su código, en cuyo caso el controlador actúa. Para evitar fragmentar el mapa de memoria se agrupan las direcciones para E/S en una zona bien definida, usualmente al principio o fin del mapa de memoria.

En la Tabla 9.16 se indican las principales ventajas e inconvenientes de ambos procedimientos. Conviene hacer notar que aunque el procesador disponga de instrucciones específicas de E/S, siempre se pueden codificar los periféricos dentro del mapa de memoria.

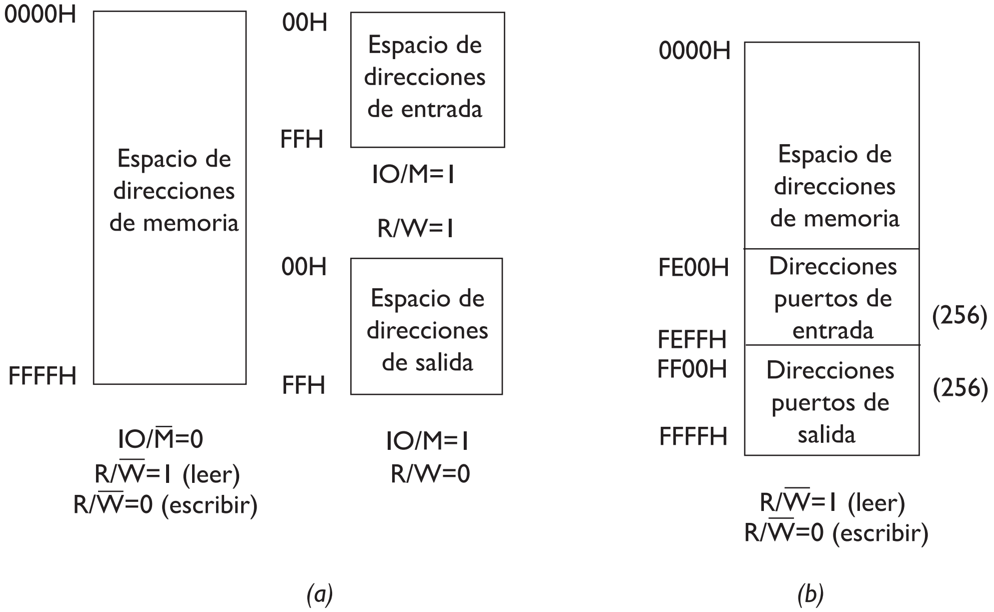


Figura 9.26. *Mapa de direcciones: (a) sistema con instrucciones específicas de E/S; (b) sistema sin instrucciones específicas de E/S.*

Tabla 9.16. *Alternativas para realizar las operaciones de entrada/salida.*

	Ventajas	Inconvenientes
Instrucciones específicas de E/S	El diseño de los circuitos de decodificación de direcciones de memoria es más simple, ya que las direcciones de memoria sólo implican a ésta. Se facilita la protección de E/S; por ejemplo, haciendo que las instrucciones IN y OUT sean privilegiadas. Los programas son más cortos y más rápidos, ya que las instrucciones de E/S son más cortas y su decodificación y ejecución más sencilla. Los programas son más legibles, ya que las instrucciones de E/S son fácilmente reconocibles.	El diseño de la CPU resulta más complejo: • Hay que incluir en la CPU circuitos específicos para codificar las instrucciones de E/S. • Hay que generar una señal de control (IO/M') para que los circuitos externos diferencien entre una operación de acceso a memoria y una E/S. El procesador debe tener un terminal más para la señal de control IO/M'.

Tabla 9.16. *Alternativas para realizar las operaciones de entrada/salida. (Cont.)*

	Ventajas	Inconvenientes
Direccionamiento de puertos dentro del mapa de memoria	Menor complejidad en el diseño de la CPU (el número de instrucciones del repertorio es menor).	Cada puerto inutiliza una posición de memoria. Las instrucciones de acceso a memoria ocupan más espacio que las específicas de E/S (en los microprocesadores de 8 bits, por ejemplo, 3 Bytes frente a 2 Bytes): • Disminuye la velocidad de procesamiento. • Aumentan los requerimientos de memoria.

9.5.3 Organización de E/S en CODE-2

Según se indicó al describir el repertorio de instrucciones de CODE-2, éste dispone de instrucciones específicas de entrada/salida, coincidiendo su mapa de direcciones con el de la Figura 9.26a.

Los puertos de E/S de CODE-2 siguen el esquema de la Figura 9.24. Por ejemplo, el puerto de salida 2 (*OP2*) está constituido por un registro como el indicado, pero de 16 bits (Figura 9.27). Las salidas de este registro *Q(15:0)* se conectan a cuatro circuitos decodificadores. Cada uno de éstos (véase Ejercicio 5.28) tiene cuatro entradas y siete salidas. Las salidas están preparadas para actuar sobre indicadores de siete segmentos (Sección 11.3.1), y activan los segmentos de forma tal que si la entrada es *0001* se visualiza un *1*, si es *0010* se visualiza un *2*, ..., si es *1110* se visualiza una *E*, y si es *1111* se visualiza una *F*. En definitiva, se muestra el valor hexadecimal del contenido del puerto.

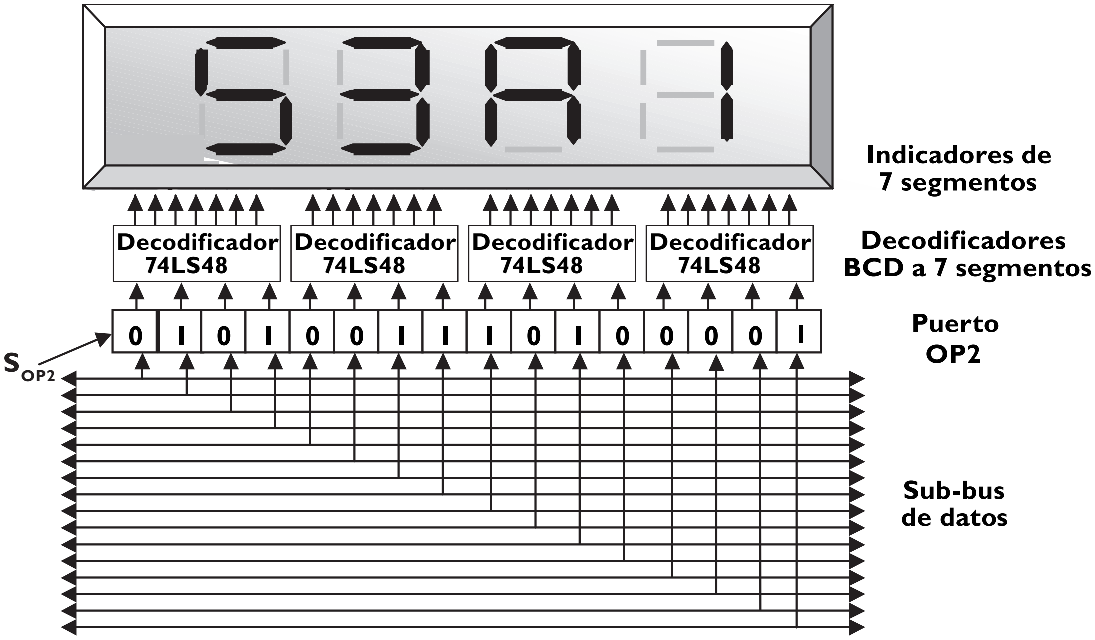


Figura 9.27. *Diseño del visualizador de contenido (OP2) de CODE-2.*

El diseño de este dispositivo de salida se efectúa de la siguiente manera (Figura 9.27):

- Las entradas *D(15:0)* se conectan directamente al sub-bus de datos, *DB(15:0).*
- Como las salidas del puerto actúan sólo y directamente sobre los circuitos adaptadores, las salidas del puerto pueden permanecer constantemente en estado de baja impedancia, por lo que hacemos:

$$T_{O2} = 1$$

- El puerto OP2 debe seleccionarse (véase Tabla 9.14) cuando:

$$R/\overline{W} = 0 \text{ e } IO/\overline{M} = 1$$

y en el bus de direcciones aparezca la dirección 2; es decir:

$$AB(7) = 0,\ AB(6) = 0,\ AB(5) = 0,\ AB(4) = 0,\ AB(3) = 0,\ AB(2) = 0,\ AB(1) = 1,\ AB(0) = 0$$

con lo que la función de conmutación que genera la señal de carga del puerto OP2 debe ser:

$$S_{OP2} = \overline{R} \cdot IO \cdot \overline{AB(7)} \cdot \overline{AB(6)} \cdot \overline{AB(5)} \cdot \overline{AB(4)} \cdot \overline{AB(3)} \cdot \overline{AB(2)} \cdot \overline{AB(1)} \cdot \overline{AB(0)}$$

obsérvese que las variables de la señal anterior se encuentran disponibles en el sub-bus de control y en el sub-bus de direcciones.

El diseño de la Figura 9.27 muestra claramente que el puerto *OP2* se cargará con el dato que hubiese en el sub-bus de datos, cuando $S_{O2} = 1$; es decir, cuando haya una operación de salida ($R/\overline{W} = 0$ e $IO/\overline{M} = 1$) y la dirección en el sub-bus de datos sea la 2.

9.5.4 Técnicas de transferencias de E/S

Hay fundamentalmente tres métodos para establecer y supervisar las operaciones de transferencias de datos de E/S: E/S programada, E/S controlada por interrupciones y E/S con controlador de acceso directo a memoria (DMA). En la Tabla 9.17 se presenta un resumen comparativo de los tres métodos, que se analizan con más detalles a continuación.

9.5.4.1 Control por medio de E/S programada

En la **E/S programada** el inicio de la operación se efectúa a iniciativa de la ejecución por el procesador de instrucciones de un programa. Existen dos variantes: E/S programada sin, o con consulta de estado.

E/S programada sin consulta de estado o incondicional

El programa, por medio del procesador, decide el momento en que se realiza la transferencia, por lo que el dispositivo de E/S debe estar siempre dispuesto a recibir datos (en el caso de una salida) o debe tener siempre datos disponibles (en el caso de una entrada). En la Tabla 9.17 se incluyen las principales características de esta técnica. Las salidas *OP1* y *OP2* de CODE-2 (indicadores luminosos) son de este tipo.

E/S programada con consulta de estado

El puerto de datos tiene asociado un puerto de estado (o de estado/control). Antes de realizar una operación de E/S de un dato de un determinado puerto, el programa incluye instrucciones para leer el puerto de estado, y comprueba si está listo o no para realizar la transferencia. Obviamente en el puerto de estado existirá un bit dedicado a almacenar si el periférico está preparado para efectuar la operación de salida (si está conectado, si la unidad de disco está cerrada, etc.), o, en el caso de una entrada, si en el puerto de entrada de datos se encuentra el dato listo para ser transferido. Obsérvese que la iniciativa de la transferencia la sigue teniendo el programa, pero

para transmitir cada dato es necesario leer los bits de estado. Con este método se adapta la velocidad de transferencia del procesador a la del periférico.

Hay dos formas de efectuar la consulta de estado: con bloqueo de programa y con consulta periódica. La consulta de estado **con bloqueo de programa**, implica que cuando hay que hacer una operación de entrada o salida y el periférico no está preparado, el programa entra en un lazo de espera hasta que el dispositivo llegue a la situación de "listo". En el procedimiento **con consulta periódica** o de **sondeo**, periódicamente se consulta el estado del periférico (o conjunto de periféricos) para ver si está preparado para recibir datos o si tiene datos para el procesador. En caso de no ser así, el programa pasa a realizar otras operaciones que no necesiten del dato a captar (caso de una entrada).

Tabla 9.17. *Resumen de las tres técnicas de control de E/S.*

	Entradas/salidas programadas	Control por medio de interrupciones	Acceso directo a memoria (DMA)
Iniciación de la operación de E/S	Por parte del programa en ejecución *(iniciación software)*	Por petición de interrupción *(iniciación hardware)*	El periférico solicita la transferencia al controlador de DMA, y éste hace a la CPU una petición de interrupción *(iniciación hardware)*
Control de la transferencia de datos	Por un programa de transferencia de datos *(control software)*	La rutina de servicio de interrupción lanza a ejecución un programa de transferencia de datos *(control software)*	La rutina de servicio de la interrupción inicializa el controlador DMA, y éste controla la transferencia de los datos *(control hardware)*
Conclusión de la operación de E/S	Finalización del programa de transferencia de datos	Finalización del programa de transferencia de datos	Un contador del controlador del DMA llega a 0. Se solicita una interrupción a la CPU.
Complejidad hardware	Mínima	Baja	Grande
Tiempo de respuesta a la petición de E/S	Grande	Rápido, o muy bajo, o pequeño	Rápido, o muy bajo, o pequeño
Velocidad de transferencia	Baja / media	Baja / media	Muy alta. Transmisión de bloques de información directamente de memoria a periférico (o viceversa) y por ráfagas
Uso de mayor interés	Periféricos muy sencillos y lentos (visualizadores, relés, etc.)	Periféricos lentos (teclado, por ejemplo) y relativamente rápidos	Periféricos muy rápidos (discos, controlador de video,..) con los que se transfieren bloques de datos.

En la Figura 9.28 se incluyen dos organigramas de programas de E/S. El primero de ellos (Figura 9.28a) corresponde a la lectura de un único periférico con bloqueo de programa, mientras el segundo (Figura 9.28b) al caso de atención de varios periféricos por el procedimiento de sondeo periódico. Obsérvese, en este último caso, que el puerto 8 es un registro de estado para diversos dispositivos, y es el propio programa quien establece la prioridad de atención de los

diversos periféricos (primero se atiende al 7, luego al 6, etc.). En la Tabla 9.17 se incluyen las principales características de esta técnica. Existe una gran variedad de circuitos integrados en el mercado que actúan como controladores, la mayor parte de ellos programables, para actuar como controladores de entrada o salida, etc. (ejemplos: chips *8155* y *8255*).

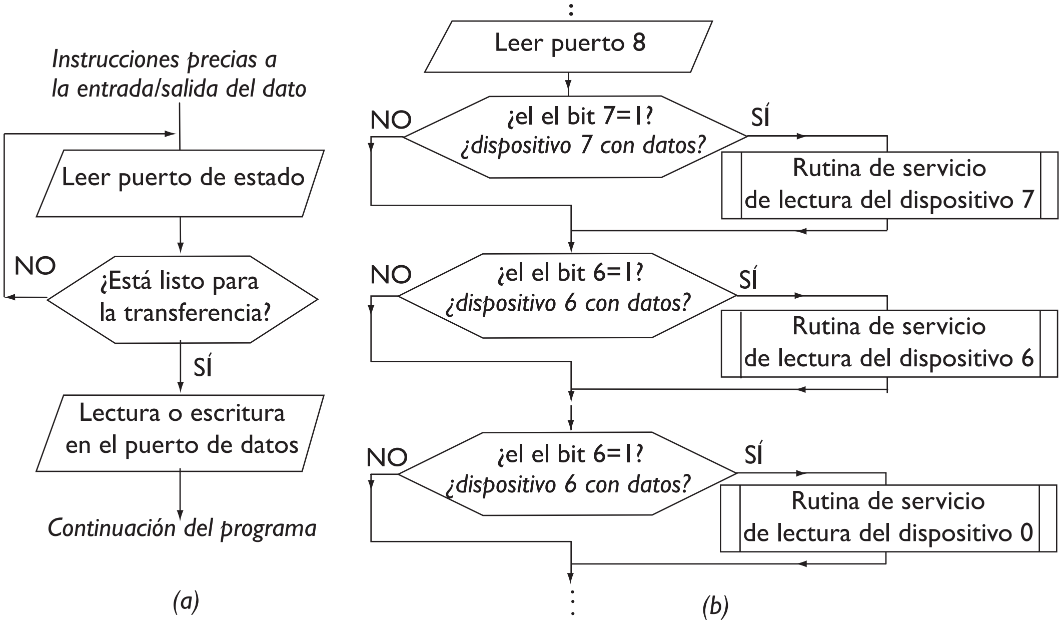


Figura 9.28. *Organigramas de algoritmos de E/S programada: (a) consulta de estado con bloqueo; (b) consulta periódica de estado sin bloqueo.*

9.5.4.2 Control por medio de interrupciones

En el método de **E/S controlada por interrupciones**, el periférico toma la iniciativa y solicita ser atendido, activando una línea de petición de interrupción. En las salidas, el periférico puede provocar una petición de interrupción si no está listo para realizar la transferencia. Cuando el procesador reconoce la interrupción, suspende el programa en curso y ejecuta un programa de gestión de interrupción para transferir información con los puertos del controlador del periférico en cuestión. Como en el caso de E/S programada, los pasos de transferencia de datos quedan bajo el control de un programa, en este caso, del programa gestor de la interrupción.

La gestión de una interrupción provocada por un periférico sigue la regla general de gestión de interrupciones vista en la Sección 7.4. Aquí la rutina de gestión de la interrupción debe identificar el periférico que solicitó la interrupción (caso de que varios de ellos compartan la misma línea de petición de interrupción) y debe efectuar la transferencia de un dato o de un bloque de datos.

Básicamente se plantean tres problemas para atender una interrupción: determinación de la dirección de la rutina de servicio de la interrupción, identificación del origen de la interrupción y gestión de prioridades de peticiones, ya que en un momento dado pueden existir varias peticiones de interrupción sin haber sido atendidas. Un computador, por lo general, tiene conectados una gran variedad de dispositivos con poder de interrupción, y la acción requerida dependerá de su causa. Es por tanto necesario identificar el origen o dispositivo que produjo la interrupción.

Otro problema que se suele plantear es la gestión de **interrupciones simultáneas**. Según se indicó en la Sección 7.4, cuando se solicita una interrupción, ésta no se acepta hasta que la instrucción (o ciclo de bus) que esté en ejecución termine. Durante ese breve tiempo pueden haberse generado otras interrupciones, que requieren ser atendidas. El procedimiento de prioridad debe determinar qué instrucción se atiende primero. Básicamente se suelen considerar cuatro tipos de soluciones al problema planteado:

1) *Gestión de prioridades centralizada por el propio procesador.* Si sólo hay un dispositivo por cada línea de interrupción, el procesador acepta la solicitud que llegue por la línea de mayor prioridad. Los tres tipos de gestión de prioridades que se analizan a continuación (2, 3 y 4) corresponden al caso en que varios dispositivos comparten una única línea.
2) *Gestión de prioridades centralizada en circuitos externos.* En este caso se utilizan circuitos especiales externos al procesador que resuelven por hardware el problema. Un circuito integrado utilizable para este cometido es el ***codificador con prioridad*** *74148* (Figura 9.29a).
3) *Gestión de prioridades por sondeo o consulta (pooling).* La entrada/salida se controla por interrupción, pero la rutina de servicio lanza un programa de sondeo para ver qué dispositivo solicitó la interrupción, análogo al analizado en la técnica de E/S por programación, siendo válido el esquema de la Figura 9.28b.
4) *Gestión de prioridades por encadenamiento (daisy-chain).* El diseño del sistema de interrupciones se hace de forma que tiene mayor prioridad el dispositivo que esté "eléctricamente" más cercano al procesador. Los distintos elementos del computador (procesador, módulos de memoria, etc) y controladores de periféricos suelen insertarse en distintas ranuras del chasis del computador, y cada ranura tiene asociada una prioridad. En la Figura 9.29b puede verse un esquema que se adapta al sistema de gestión por encadenamiento.

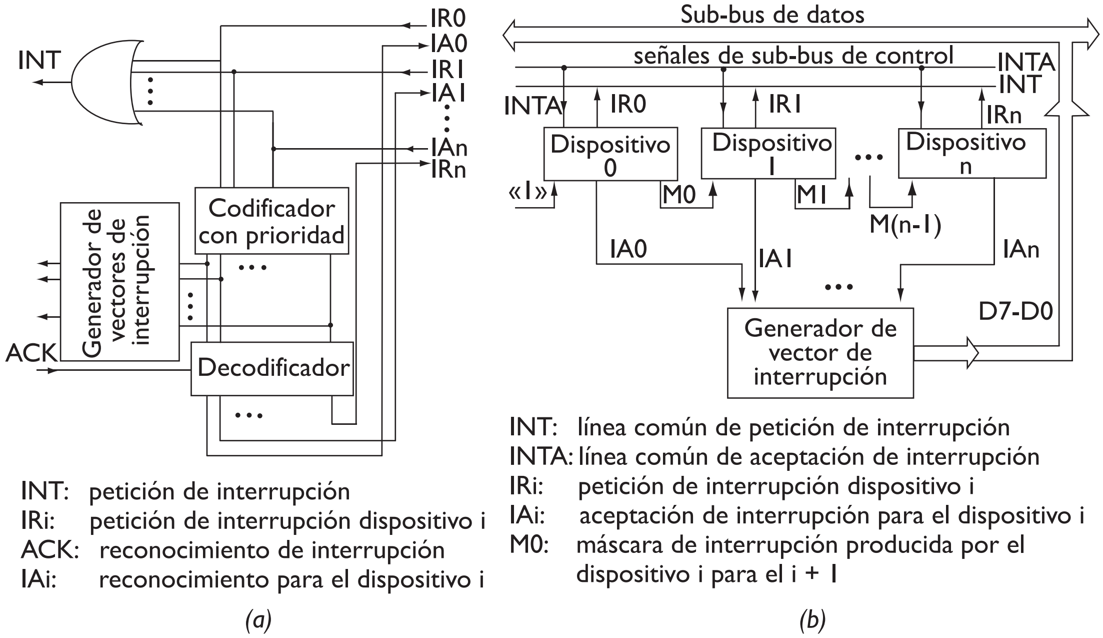


Figura 9.29. *Gestión de interrupciones: (a) centralizada con circuitos externos; (b) por encadenamiento.*

En Tabla 9.17 se resumen las principales características de esta técnica. La mayoría de las familias de microprocesadores disponían de circuitos integrados **controladores de interrupciones** para gestionar eficientemente las interrupciones; éste es el caso del Controlador de Interrupciones Programable *PIC 8259* (*Programmable Interrupt Controller*). En la actualidad los controladores se integran en los ***chipset*** asociados al microprocesador en cuestión.

9.5.4.3 Controlador de acceso directo a memoria (DMA)

Debido a lo rutinario de la programación de las operaciones de entrada/salida y a la gran frecuencia de su utilización, se ha ideado la técnica de **acceso directo a memoria** (***DMA***: *Direct Access Memory*) que permite descargar al procesador de las operaciones más elementales de E/S, y que implica la utilización de circuitos especializados (**controladores DMA** o ***CDMA***).

El CDMA actúa como controlador del bus y supervisa las transferencias de información entre memoria principal y uno o más controlador de E/S, sin intervención directa del procesador, salvo en la inicialización y final. Tanto en los casos de E/S programadas como controladas por interrupción, debe ejecutarse por el procesador una rutina gestora del periférico que controla minuciosamente la operación de entrada/salida; la ejecución de estas instrucciones es muy frecuente y consume un considerable tiempo de procesador. El controlador DMA persigue resolver este problema y con él se pueden lograr velocidades de transferencia sólo limitadas por la velocidad de los buses y los tiempos de acceso a memoria y al periférico. Este incremento de la velocidad es posible ya que la mayoría de los pasos para efectuar una E/S se realizan por hardware, mantenimiento de la dirección de memoria a acceder en un contador especializado, y es especialmente notable cuando se transfieren bloques de información entre memoria principal y un periférico y viceversa. Piénsese que ésta es la situación más corriente cuando se trata de periféricos rápidos (en un disco, por ejemplo, la cantidad mínima de transferencia de información corresponde a una unidad de asignación o *cluster*, que puede ser del orden de 512 a 2048 Bytes).

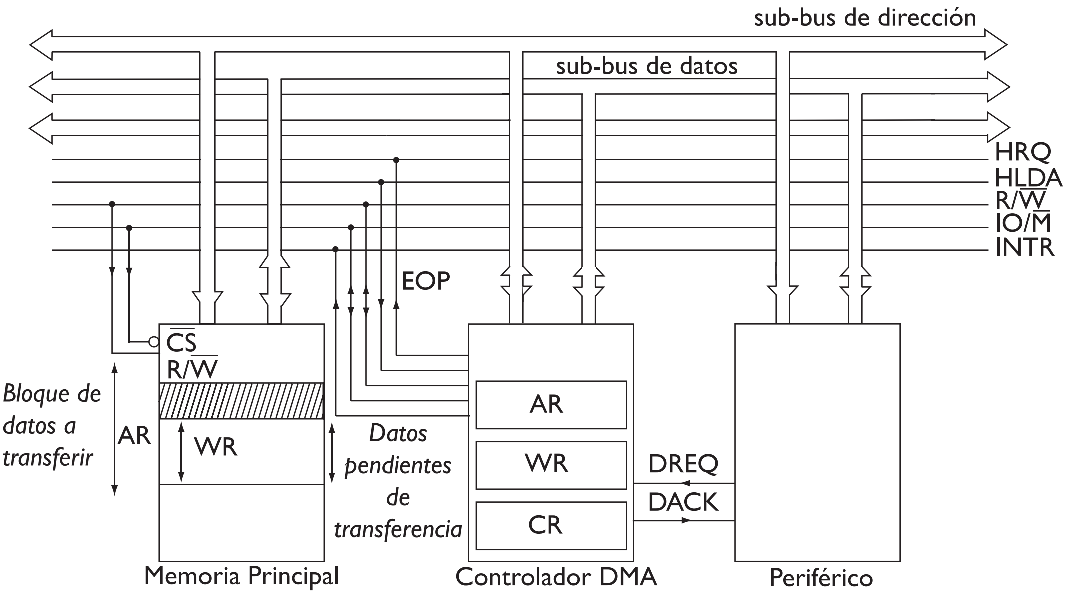
Cuando va a actuar el CDMA, el procesador le cede el control del bus del sistema (salvo algunas líneas del sub-bus de control) encargándose aquél de generar tanto las direcciones de memoria y del dispositivo implicado en la transferencia, como las señales de control ($R/\overline{W}$ e $IO/\overline{M}$) necesarias para llevar a buen término las mismas. En definitiva, el CDMA actúa como controlador o dueño del bus del sistema.

Una vez inicializado el CDMA, cada transferencia de palabra de datos se realiza en tan sólo un ciclo de escritura (o lectura) de memoria (uno o pocos ciclos de reloj). Por contra, como se ha indicado anteriormente, una E/S programada o realizada con una rutina de servicio de interrupción (ISR) necesita la ejecución de varias instrucciones (numerosos ciclos de reloj).

La técnica DMA se utiliza principalmente con periféricos rápidos, tales como discos, controladores gráficos, impresoras de alta velocidad, sistemas de adquisición de datos, etc.

En la Figura 9.30 puede verse un esquema simplificado de un CDMA y su interconexión a otros elementos del computador. En su interior contiene al menos los siguientes puertos de E/S:

- *AR:* registro que contiene la dirección de memoria implicada en la operación de entrada o salida.
- *WR:* contador que contiene el número de palabras de datos que quedan por transferir.
- *CR:* registro de control y estado; contiene información tal como sentido de la transferencia (transferencia desde o hacia el dispositivo), estado de error o correcto de la transferencia, etc.



AR:	dirección del próximo dato	HRQ:	petición del bus por el CDMA
WR:	número de palabras que quedan por transferir	HLDA:	cesión del bus por la CPU al CDMA
CR:	registro de control/estado	DREQ:	petición del CDMA por el periférico
		DACK:	atención del CDMA a la petición del periférico

Figura 9.30. *Esquema simplificado de un controlador y una posible conexión al bus del procesador.*

La inicialización del CDMA suele hacerse con una rutina que escribe en *AR* la dirección inicial de memoria donde se encuentra el bloque a transferir, en *WR* el tamaño del bloque y en *CR* la dirección de la transferencia.

Cuando el dispositivo de E/S requiere la actuación del CDMA activa la señal *DREQ* (petición por el dispositivo de E/S). Al recibir el CDMA esta señal, genera una petición del bus al procesador (activa *HRQ*) y cuando recibe la conformidad (*HLDA*) genera, para el dispositivo de E/S, la señal *DACK* (aceptación al dispositivo). En está situación se efectuaría la transferencia de datos entre dispositivo y memoria a través del bus del sistema. Obsérvese que los datos a transferir no tienen por qué pasar a través del CDMA (si se hace así se tiene una ralentización de la transferencia) sino que pueden llevarse directamente, a través del bus del sistema, de memoria a periférico, o viceversa. El controlador va proporcionando las señales de control de las transferencias y las direcciones de memoria involucradas en las mismas (que mantiene en *AR*). Al realizar cada transferencia se decrementa en 1 el valor de *WR*, y cuando llega a cero, el CDMA solicita una interrupción al procesador a través de la línea *EOP* (*fin de proceso*). La subrutina que atiende esta interrupción lee el registro *CR* (puerto de estado) para comprobar si la transferencia se ha efectuado correctamente.

Hay cuatro métodos o formas básicas de realizar el acceso directo a memoria:

1. **Transferencia de bloques o parada del procesador.** Se transmite una secuencia de palabras de datos (un *bloque*) en forma de ráfaga continua. El CDMA toma el control

del bus durante todo el período que dura la transferencia de datos (hasta que *WR = 0*). El procesador, por tanto, no tiene acceso al bus hasta que la transferencia termina, lo que obliga a esperar intervalos de tiempo que pueden llegar a ser largos. Nótese que aunque el procesador quede paralizado, la transferencia del bloque se hace más rápidamente que si no existiese CDMA: el dato va directamente de memoria a periférico –o viceversa–, sin necesidad de pasar por el acumulador del procesador u otro registro intermedio.

2. **Transferencia con robo de ciclo.** La transferencia de un bloque se realiza de dato en dato. El CDMA, de vez en cuando, *roba* al procesador uno o varios ciclos máquina durante los cuales utiliza el bus para transferir una palabra del bloque. El procesador utiliza el bus durante los ciclos no robados. Un robo de ciclo puede aceptarse en mitad de una instrucción, y hace que la duración efectiva de las instrucciones sea mayor. En la Figura 9.31 puede observarse el alargamiento en la duración de las instrucciones debido al DMA.
3. **DMA transparente.** La transferencia se realiza también palabra a palabra, pero sin interrumpir la ejecución del programa ejecutado por el procesador. Para ello el CDMA utiliza el bus del sistema sólo en aquellos ciclos en los que el procesador no lo utiliza, como pueden ser los ciclos de decodificación del código de operación o en otros que sólo se realizan operaciones internas (transferencias entre registros, operaciones aritméticas, etc.). El CDMA detecta los ciclos de no utilización del bus analizando las líneas de estado del procesador.
4. **DMA con sistemas multibús.** La máxima velocidad puede conseguirse utilizando memorias multipuerta, constituidas por varios bloques de memoria, cada uno de ellos con su registros de dirección y de lectura/escritura correspondientes. De esta forma simultáneamente pueden conectarse tantos buses como puertas haya (un bus a cada una de las puertas).

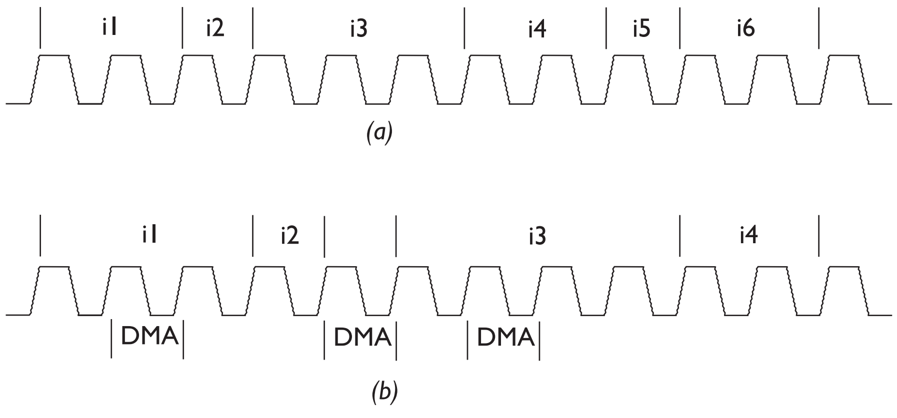


Figura 9.31. *Ejemplo de temporización en la ejecución de 6 instrucciones (i1 a i6): (a) sin utilización simultánea de DMA; (b) con DMA, ciclo robado.*

Tanto en la elección del método de transferencia de entrada/salida, como en la elección del modo de control de DMA debe llegarse a un compromiso entre:

- Velocidad de transferencia.
- Velocidad de ejecución del programa principal.
- Complejidad del hardware.

Existen diversos circuitos integrados que actúan como CDMA, tales como el *8257* y el *8237*. Es usual en este tipo de circuitos que se puedan controlar simultáneamente 4 operaciones DMA independientes. En la actualidad el controlador DMA suele incluirse dentro del ***chipset*** asociado al microprocesador y a la placa base utilizados en el sistema (Sección 12.6).

9.6 Panel de control de CODE-2

Según se ha comentado a lo largo de los Capítulos 8 y 9, CODE es un computador orientado a la enseñanza del funcionamiento y diseño de este tipo de sistemas. Para facilitar su uso y construcción los periféricos con que cuenta son muy elementales, y están todos incluidos en el mismo chasis donde se encuentran los otros elementos (procesador y memoria). Como ya indicamos en la Sección 8.3.6, el panel de control incluye seis teclas de órdenes (paso a paso, dirección, registros, cargar, ejecutar, continuar), un teclado hexadecimal asociado al puerto de entrada *IP1*, y dos puertos de salida (*OP1* y *OP2*) acoplados a indicadores de siete segmentos. La Figura 9.32 es repetición de la Figura 8.35, que volvemos incluir aquí para comodidad del lector, ya que muestra todos los elementos del panel de control.

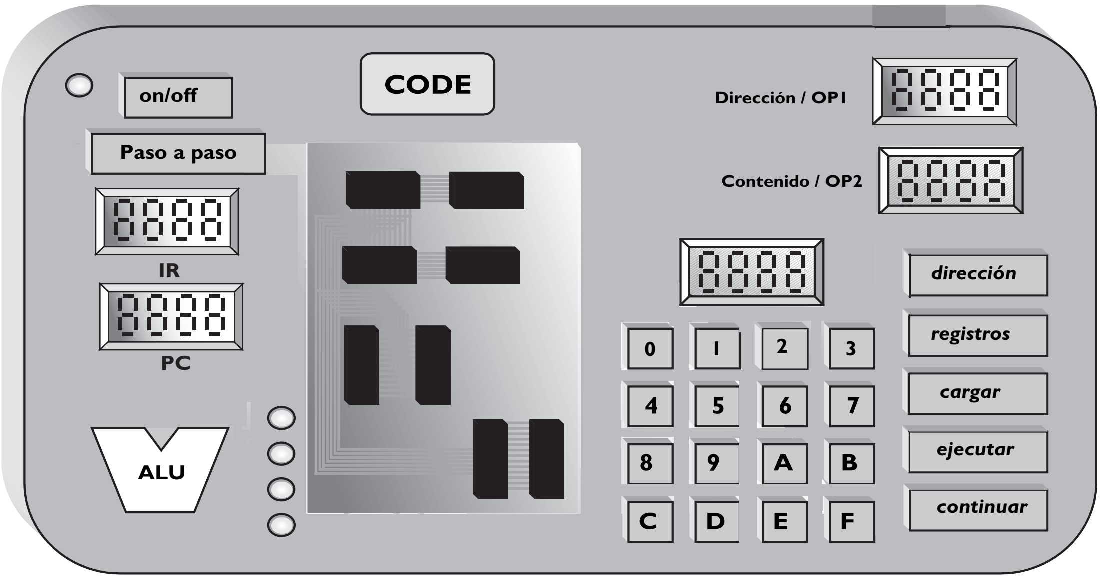


Figura 9.32. *Aspecto del panel de control de CODE-2.*

Con las teclas de control se pretende realizar determinadas operaciones tales como leer directamente contenidos de memoria, cargar datos o programas en memoria, leer el contenido de los registros o lanzar a ejecución un programa. En esta sección se describe el diseño del sistema asociado al panel de control.

Cuando se pulsa una tecla de control se activan unos biestables interiores que juntos conforman un registro que denominamos **puerto de estado/control del panel**. Uno de estos biestables es el de petición de interrupción, *ir* (de *interruption request*), y, cuando es activado, se provoca una interrupción que, según hemos visto en la Sección 9.3.1, hace una llamada a la rutina que se encuentre en la posición F800 de memoria (al inicio de la memoria ROM). El

conjunto de programas de control de CODE-2 constituye un pequeño **monitor de control,** que está almacenado en la memoria ROM. El monitor utiliza además una zona de la memoria RAM (de la posición F000 a la F800) para almacenar datos temporales que necesita para su propia actividad. La gestión de interrupciones se efectúa siguiendo el esquema de la Figura 9.33.

En primer lugar, y siguiendo el esquema general de atención de interrupciones descrito en la Sección 7.4, se salvaguardan los contenidos de todos los registros y biestables indicadores de la ALU en la memoria principal (zona a partir de la posición *F000*). Posteriormente se lee el contenido de los biestables del puerto de estado/control para identificar el origen de la interrupción. Una vez detectado el origen, pone a cero el biestable correspondiente y hace una llamada a la rutina que gestiona el servicio requerido (carga de programa, leer una zona de memoria, etc.). Una vez realizado el servicio, se restauran los contenidos iniciales de los registros (que previamente se habían salvado en memoria) y por último se incluye una instrucción de retorno, para que continúe ejecutándose el programa interrumpido.

Hay que hacer notar que siempre que se produce un servicio de interrupción, el monitor debe dejar todos los elementos del procesador exactamente como estaban antes de iniciarse la atención del servicio. La única excepción es el contenido del registro *rF,* que tal y como se indicó en la Sección 8.3.6, no es conservado o restituido por el monitor.

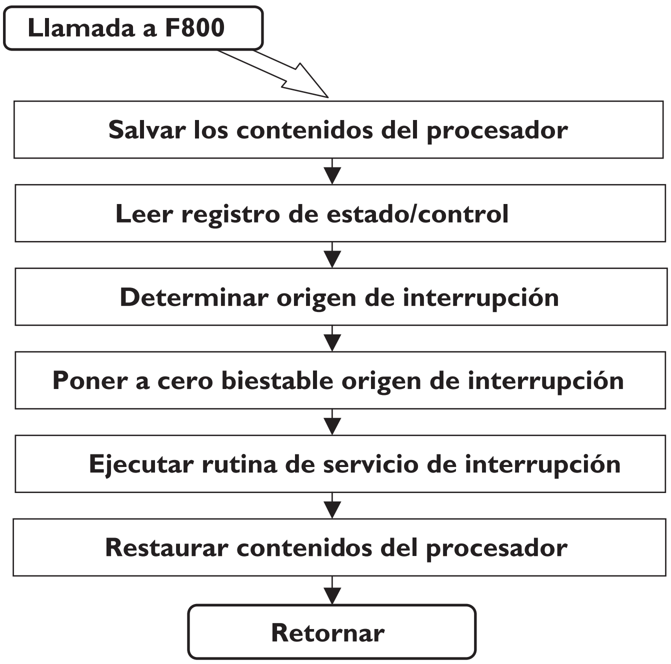


Figura 9.33. *Esquema de gestión de interrupciones en CODE-2.*

9.6.1 Puerto de estado/control del panel

Para entender adecuadamente el funcionamiento del monitor del panel de control conviene describir los elementos del puerto de estado/control. Este puerto está constituido por 8 biestables (Figura 9.34), con las siguientes denominaciones y cometidos:

- Biestable **set**. Se activa (pone a 1) cuando se pulsa la tecla de conexión de energía eléctrica (**on/off**).

- Biestable **esp**. Se activa en todos los estados *c1*, siempre y cuando está conectado el interruptor **paso a paso**, y siempre que haya que entrar en modo de espera, tal como en la ejecución de la instrucción máquina *HALT*.
- Biestable **ir**. Biestable de petición de interrupción. Se activa al conectar CODE (**on/off**), y pulsar cualquiera de las siguientes teclas de órdenes: **dirección** o **registro** o **ejecutar**.
- Biestable **dcc**. Se activa al pulsar la tecla **dirección**.
- Biestable **rtr**. Se activa al pulsar la tecla **registros**.
- Biestable **car**. Se activa al pulsar la tecla **cargar.**
- Biestable **ejc**. Se activa al pulsar la tecla **ejecutar**.
- Biestable **con**. Se activa al pulsar la tecla **continuar**.

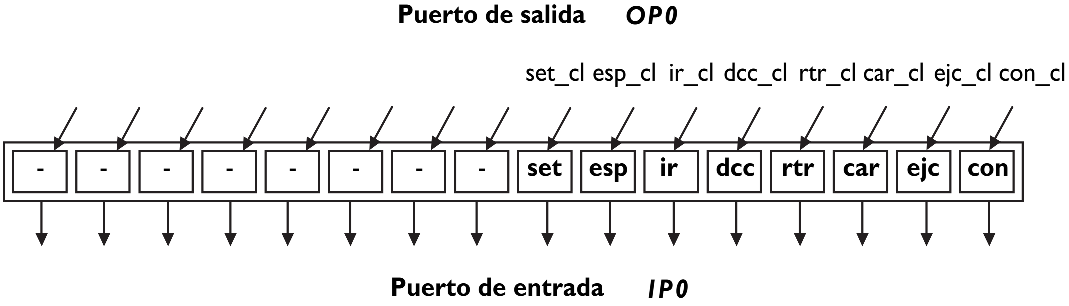


Figura 9.34. *Puerto de estado/control.*

En resumen, el panel se ha diseñado de forma que al pulsar las distintas teclas de órdenes se activan los biestables de la forma que a continuación se indica:

- **On/off**, se activa **set** e **ir**.
- **Paso a paso**, se activa **esp**.
- **Dirección**, se activan **dcc** e **ir**.
- **Registros**, se activan **rtr** e **ir**.
- **Cargar**, se activan **car** e **ir**.
- **Ejecutar**, se activan **ejc** e **ir**.

El puerto de estado se ha configurado como puerto de entrada 0 (*IP0*), de forma que se puede leer en su totalidad, con una instrucción de entrada. A su vez, las entradas asíncronas de puesta a cero de los biestables se han dispuesto como puerto de salida 0 (*OP0*), de esta forma desde un programa (y en particular desde el monitor) es posible desactivar (poner a cero) selectivamente lo distintos biestables con instrucciones de salida (*OUT*):

- Desactivar **set**: escribir *H'0080* en *OP0.*
- Desactivar **esp**: escribir *H'0040* en *OP0.*
- Desactivar **ir**: escribir *H'0020* en *OP0.*
- Desactivar **dcc**: escribir *H'0010* en *OP0.*
- Desactivar **rtr**: escribir *H'0008* en *OP0.*
- Desactivar **car**: escribir *H'0004* en *OP0.*
- Desactivar **ejc**: escribir *H'0002* en *OP0.*
- Desactivar **con**: escribir *H'0001* en *OP0.*

9.6.2 Monitor de CODE-2

Como se ha indicado anteriormente, el monitor se encuentra grabado en la memoria ROM, y utiliza la zona más alta de la memoria RAM para almacenar sus datos.

Recuérdese (Sección 9.3.1 y Figura 9.12) que cuando al iniciarse la ejecución de una instrucción (estados *c0*) el biestable de petición de interrupción está activo (*ir* = *1*), se ejecuta una llamada a la rutina que empieza en la dirección F800; es decir, al monitor.

El monitor sencillamente es un conjunto de módulos que implementan las acciones indicadas en la Figura 9.33. Los contenidos de los registros del procesador se salvan en la zona de memoria RAM reservada para el monitor. La lectura del registro de estado/control se efectúa sencillamente con una instrucción *IN rx,IP0;* es decir, leyendo el contenido del puerto de entrada *IP0*. Por medio de desplazamientos, o utilizando cualquier otra técnica de las descritas en la Sección 8.3.3, se puede acceder al contenido de cada bit (biestable) específico; pudiéndose así identificar el origen de la interrupción.

Una vez detectado el origen de la interrupción se puede desactivar el biestable correspondiente con una instrucción *OUT OP0,rx*, y, posteriormente producir un salto al módulo que controla la acción requerida (cargar un programa, leer una zona de memoria, etc.). Estos módulos, en general, acaban llamando a otro módulo que restaura los contenidos iniciales del procesador (registros e indicadores de estado de la ALU), y produciendo el retorno al programa interrumpido.

Los principales módulos de que consta el monitor son:

- Servicio de salvaguarda de contenidos del procesador.
- Servicio de restauración de contenidos del procesador.
- Servicio de identificación de origen de interrupción.
- Servicio de arranque.
- Servicio de acceso a memoria, y
- Servicio de acceso a registros.

En la Sección 8.3.6 se describieron las tareas asociadas a cada tecla de órdenes, y que por tanto definen las funciones que deben realizar los distintos servicios.

En la página web de este texto se incluye una descripción más detallada sobre el monitor, que puede ser consultada por el lector interesado en él.

9.7 Conclusión

En este capítulo hemos presentado las distintas unidades funcionales que constituyen un computador von Neumann en el nivel de micromáquina. Después de exponer algunas ideas sobre la aritmética del computador, hemos analizado los objetivos y estructura de la unidad de tratamiento, unidad de control y memoria. Por último hemos presentado los conceptos básicos de la organización de las entradas/salidas (que es uno de los aspectos más complejos de la estructura de computadores). Para clarificar las ideas expuestas se ha efectuado un análisis sobre la estructura del computador didáctico elemental CODE-2. Se ha efectuado su diseño a nivel de transferencia entre registros (RTL), incluyendo dos metodologías para el diseño de la unidad de control: cableada y microprogramada.

EJERCICIOS

9.1 Suponer un computador en el que los números enteros ocupan n = 8 bits, y se representan y operan en complemento a 1. Reproducir los cálculos que se indican a continuación *(no olvidar comprobar si se produce desbordamiento –"overflow"– o no).*
a) 1010 1011 – 0101 1011
b) 0111 0001 – 0101 1001
c) 1010 1011 + 1110 0111
d) 0101 0100 – 0001 1000

9.2 Realizar en binario los cálculos que a continuación se indican suponiendo que el bit de la izquierda es el signo, y que se utiliza la representación en complemento a 1.
a) 82 – BE
b) 74 –A6
c) C5 + D4
d) 58 + A8

9.3 Un computador representa los números enteros con 8 bits y en complemento a 2. Efectuar las dos siguientes operaciones aritméticas (especificar si hay desbordamiento o no y el valor del resultado en binario, hexadecimal y decimal):
a) 96 – 6E
b) C6 + 3D

9.4 El contenido en hexadecimal de dos datos, en precisión sencilla, representado en notación IEEE 754, es:
X = DEB0 0000 Y = 5DE0 0000
Reproducir las operaciones que efectuaría el computador (en binario o hexadecimal) para obtener:
a) X + Y
b) X – Y
c) X · Y
d) X/Y
¿Qué número decimal representa X?

9.5 Sea el número $A = -49{,}7432 \cdot 10^{-7}$
a) ¿Cuál es la representación interna en simple precisión según las especificaciones IEEE 754? *(Dar el resultado en hexadecimal empaquetado.)*
b) Si otro dato, B, también en simple precisión IEEE 754, en hexadecimal empaquetado es *B = 59F3 215F*. ¿Cómo calcularía internamente y que resultado obtendría el computador al efectuar A + B?
c) ¿Qué valores puede tomar el exponente de B para que A + B = A?

9.6 En la memoria de CODE-2, a partir de la dirección 01A6 se encuentran las siguientes instrucciones:
1ª LLI rD,H'A2
2ª LD r0,H'05
3ª SUBS r0,r0,rD
4ª OUT OP2,r0
a) Indicar, en una tabla, el número total de accesos a memoria que se realizan en la ejecución completa de cada una de las instrucciones, así como los contenidos (en hexadecimal) de PC, IR, DR, AR, r0, rD y OP2 después de la captación del codop (final del estado *c4)* y después de la ejecución de cada una de las 4 instrucciones. (*Sugerencia:* pasar las instrucciones a código máquina.)
b) Si el reloj de CODE funcionase a una frecuencia de 1 GHz, ¿cuánto se tardarían en ejecutar las instrucciones anteriores?

9.7 Se desea modificar CODE de la siguiente forma:
a) Se eliminan las instrucciones de entrada (IN) y Salida (OUT), para incluir los puertos de entrada y salida en el mapa de memoria.
b) Se aprovechan los códigos de operación liberados para incluir las siguientes 2 nuevas instrucciones:

Codop	*Nemónico*	*Significado*
0100	INR rx	A ← A + 1
0101	DCR rx	A ← A – 1

a) Indicar las modificaciones que habría que realizar en la unidad de control cableada de CODE para implementar los cambios deseados.
b) Indicar las modificaciones que habría que realizar en la unidad de control microprogramada de CODE para implementar los cambios deseados.
c) Detalle cómo podrían realizarse las operaciones de entrada y salida al haber suprimido instrucciones específicas para hacerlas.

9.8 Diseñar una memoria RAM de 16 MBytes organizada en palabras de 32 bits con el circuito integrado IS61C6416 descrito en la Sección 9.4.

9.9 Diseñar una memoria ROM de 64 KBytes organizada en palabras de 32 bits con el circuito integrado 27C64 descrito en la Sección 9.4.

9.10 Un teclado compuesto de 64 teclas se conecta a los puertos de entrada y de salida de CODE tal como se indica en la Figura 9.35. Para detectar

una tecla pulsada el programa de lectura del teclado genera la secuencia:

1000 0000
0100 0000
0010 0000
0001 0000
0000 1000
0000 0100
0000 0010
0000 0001

(esta secuencia se denomina *paseo del 1*) que va escribiendo sucesivamente en el puerto de salida *OP3*. Los bits de salida, una vez almacenados en el puerto, se aplican directamente a las 8 columnas de la matriz del teclado, y los valores de tensión que toman las filas se almacenan directamente en el puerto de entrada IP3, donde son almacenadas para sucesivas lecturas por el procesador (se escribe una palabra de la secuencia en el *OP3*, se lee lo almacenado en *IP3*, se escribe la siguiente palabra de la secuencia en *OP3*, se lee *IP3*, y así sucesivamente). Si se pulsa cualquier tecla, pasará un 1 de la correspondiente columna (Y) a la correspondiente fila (X), en el instante que esté activada por la palabra de la secuencia correspondiente a dicha columna. Comparando el valor leído en *IP3* con la palabra almacenada inmediatamente antes en OP3, puede identificarse la tecla pulsada (X,Y), llamando a las teclas *11, 12, 13, 14, 15, 16, 17, 18, 21, ..., 88*. Realizar una subrutina que almacene en la dirección *7FFF* el código de la tecla pulsada.

9.11 Un módulo DMA transfiere, mediante el procedimiento de *robo de ciclo,* caracteres a la memoria desde un periférico a una velocidad de 9.600 bps (bits/segundo). El procesador, de 32 bits, capta instrucciones a una velocidad media de 1 millón de instrucciones por segundo (1 MIPS). ¿Cuál será la velocidad media del procesador teniendo en cuenta el efecto del DMA? ¿Qué otros procedimientos de transferencia a través de DMA mejorarían la velocidad efectiva del procesador?

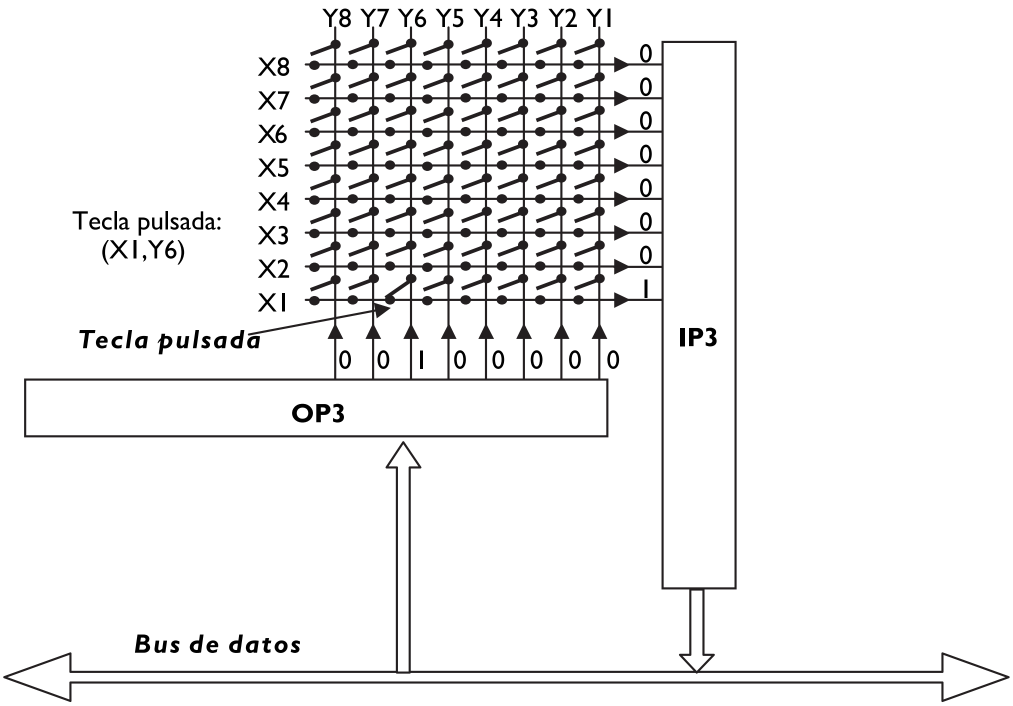


Figura 9.35.

BIBLIOGRAFÍA

[Bou82] Boulaye, G: *Microprogramación,* Marcombo. 1982.

[Car01] Carpinelli, J. D.: *Computer Systems. Organization & Architecture,* Addison Wesley, 2001.

[Dia02] Díaz-Alonso, J.; Román, D. C.; del Pino, B.; Gómez Mula, F.; Prieto, A.; Pelayo, F. J.: *Implementación del computador CODE-2*, Actas de las II Jornadas sobre Computación Reconfigurable y Aplicaciones (JCRA'2002), pp. 249-256, Almuñecar (Granada), 18-20 septiembre 2002.

[Fer98] Fernández, G.: *Conceptos básicos de arquitectura y sistemas operativos;* Sistemas y Servicios de Comunicación, 1998.

[Flo06] Floyd, T. L.: *Digital Fundamentals*, 9th Edc., Prentice Hall, 2006.

[Gaj92] Gajski, D.; Dutt, N. D.; Wu, A. C-H.; Lin, S. Y-L.: *High-level synthesis. Introduction to chip and systems design;* Kluwer, 1992.

[Gaj97] Gajski, D.: *Principios de diseño digital;* Prentice Hall, 1997.

[Ham03] Hamacher, C.; Vranesic, Z.; Zaky, S.: *Organización de computadores*, 5ª Ed., MacGraw-Hill, 2003.

[Hay98] Hayes, J. P.: *Computer Architecture and Organization,* 3ª Edc. McGraw-Hill, 1998.

[Hen93] Hennessy, J. L.; Patterson,D.: *Arquitectura de computadores. Un enfoque cuantitativo*, McGraw-Hill, 1993.

[Her98] Hermida, R.; Corral, A. M. del; Pastor, E.; Sánchez, F.: *Fundamentos de computadores,* Síntesis, 1998.

[Hwa93] Hwang, K.: *Advanced Computer Architecture,* McGraw-Hill, 1993.

[Kor93] Koren, I.: *Computer Arithmetic Algorithms,* Prentice Hall, 1993.

[Llo03] Lloris, A.; Prieto, A.; Parrilla, L.: *Sistemas Digitales,* McGraw-Hill, 2003.

[Man03] Mano, M.; Kime, C.: *Logic and computer design fundamentals,* Prentice Hall, 2003.

[Mig04] Miguel, P. de: *Fundamentos de los computadores,* 9ª Edc. Thomson, 2004.

[Mur00] Murdocca, M. J.; Heuring, V. P.: *Principles of Computer Architecture,* Prentice Hall, 2000.

[Omo94] Omondi, A.: *Computer arithmetic systems: Algorithms, Architecture and Implementations,* Prentice Hall, 1994.

[Pat00] Patterson, D. A.; Hennessy, J. L.: *Estructura y diseño de computadores,* Editorial Reverté, 2000.

[Pat01] Patt, Y. N.: *Introduction to computing systems,* McGraw-Hill, 2001.

[Raf90] Rafiquzzaman, M.; Chandra, R.: *Arquitectura de Ordenadores,* Anaya Multimedia. 1990.

[Ruz97] Ruz, J.: *De la tecnología a la arquitectura de computadores,* Síntesis, 1997.

[Sim97] Sima, D.; Fountain,T.; Kacsuk, P.: *Advanced Computer Architectures: A design space approach,* Addison-Wesley, 1997.

[Sta06] Stallings, W.: *Organización y arquitectura de computadores*, 7ª Edc. Prentice Hall, 2006.

[Swa90] Swartzlander, E. (Edt.): *Computer Arthmetic,* (Volumen 1 y 2), IEEE Computer Society Press, 1990.

[Tan00] Tanenbaum, A.S.: *Organización de computadoras. Un enfoque estructurado,* 4ª Edc. Pearson Educación, 2000.

[Ter98] Terás, Ll.; Torroja, Y.; Locos, S.; Villar,E.: *VHDL. Lenguaje estándar de diseño electrónico,* McGraw-Hill, 1998.

[Wal92] Wallis, P.: *Improving Floating-Point Programming,* Wiley, 1992.

[Wil94] Wilkinson, B.: *Computer architecture: design and performances,* Prentice Hall, 1994.

[Zar96] Zarghan, M. R.: *Computer Architecture,* Prentice Hall, 1996.

CAPÍTULO 10

El sistema de memoria

En este capítulo presentamos distintos aspectos acerca del sistema de memoria de un computador. En primer lugar (Sección 10.1), se analiza el concepto de jerarquía de memoria que pretende dar una visión global y unificada de distintos medios de memorización, diferenciados fundamentalmente por su capacidad de almacenamiento y su velocidad de funcionamiento. Posteriormente, se estudia la memoria interna (Sección 10.2), constituida básicamente por la memoria caché y la memoria principal. En último lugar (Sección 10.3) se describen los principales dispositivos de memoria externa que pretenden soslayar los problemas de la memoria principal en cuanto a su limitada capacidad y pérdida de información al desconectarla de su fuente de energía eléctrica. Los dispositivos de memoria externa que se discuten son: discos y cintas magnéticas, discos ópticos y memoria flash USB.

10.1 Jerarquía de memoria

El procesador capta las instrucciones y datos de la memoria principal, almacenando en ella los resultados de las operaciones; por tanto, es conveniente que las velocidades de funcionamiento de ambas unidades sean del mismo orden de magnitud; sin embargo, esto no suele ser así, según se pone de manifiesto en el siguiente ejemplo.

EJEMPLO 10.1

Suponiendo un microprocesador Pentium IV con frecuencia de reloj de 1,3 GHz y que ejecute por término medio 4 instrucciones/ciclo; esto quiere decir que ejecuta $1{,}3 \times 10^9$ ciclos/s · 4 instrucciones/ciclo $= 5{,}2 \times 10^9$ instrucciones/segundo.

Bajo la hipótesis de que se capta la instrucción completa en cada acceso a memoria, el resultado anterior implica que habría que efectuar $5 \times 2 \times 10^9$ accesos/segundo, o lo que es lo mismo, el tiempo de acceso tendría que ser de $(5 \times 2 \times 10^9)^{-1} = 0{,}19$ ns.

No obstante, las memorias RAM dinámicas (que son las que se utilizan para diseñar la memoria principal) tienen tiempos de acceso del orden de 50 ns, con lo que son unas 250 veces más lentas que la velocidad requerida. Hay un salto de 3 órdenes de magnitud entre las velocidades del procesador y de la memoria principal.

(Nota: este ejemplo es únicamente ilustrativo, ya que como se verá en la Sección 10.2.2 si las instrucciones se encuentran consecutivamente en memoria se pueden obtener tiempos de acceso efectivos mucho menores.)

Para paliar este problema, se utiliza una memoria especial denominada **memoria caché** (a veces denominada también **antememoria** o **memoria oculta**) que se coloca entre la memoria principal y el procesador, de forma tal que éste se comunica directamente con ella y no con la memoria principal, siendo *transparente* al programador ya que el computador actúa funcionalmente como si no existiese. Como veremos en la Sección 10.2.1 la memoria caché suele estar constituida por circuitos integrados SRAM que son más rápidos (aproximadamente 8 y 16 veces) que los DRAM que se utilizan para la memoria principal.

La memoria principal no tiene un tamaño (capacidad) suficiente para atender las necesidades de almacenamiento de las aplicaciones y de los usuarios, por lo que es necesario disponer de otros elementos más capaces (discos magnéticos y ópticos, y cintas magnéticas) pero que, desafortunadamente, son mucho más lentos (unas 200.000 veces, véase Tabla 10.1); con lo cual, como en toda transferencia de información entre dos unidades la unidad más lenta impone la velocidad, el rendimiento global del computador sería muy bajo.

Tabla 10.1. *Órdenes de magnitud comparativos de varios parámetros de distintas unidades de un computador.*

Nivel	Dispositivo	Capacidad	Tiempo de acceso	Ancho de banda MB/s
0	Registros procesador	6, 8, 16, ..., 256	0,25 a 0,5 ns	20.000 a 100.000
1	Memoria caché (SRAM)	8 KB a 8 MB	0,5 a 25 ns	5.000 a 10.000
2	Memoria principal (DRAM)	1 MB a 32 GB	80 a 200 ns	1.000 a 5.000
3	Disco duro (magnético)	5 GB-1,2 TB	3 a 15 ms	3 a 320
4	Disco óptico CD y DVD	680 MB	80 a 220 ms	0.6
7	Cinta magnética	300 KB a 800 GB	60 a 160 s	1 a 30

Considerando globalmente el sistema de memoria para almacenamiento de información de un computador, se puede establecer una **jerarquía de memoria** (Figura 10.1), que hay que considerar bajo cuatro puntos de vista:

a) tamaño o capacidad, s, de almacenamiento, suficiente,
b) tiempo de acceso, t, lo menor posible,
c) ancho de banda, b, alto y
d) coste por bit, c, reducido.

Desgraciadamente no existe una tecnología concreta que reúna simultáneamente las cuatro características indicadas. Debido a ello se han desarrollado múltiples medios de almacenamiento que se discuten en el presente capítulo. En la Tabla 10.1 se incluye un resumen de las características de los principales dispositivos de memoria. En el nivel superior (nivel 0) se encuentran los registros internos de la unidad de procesamiento, y en el inferior las cintas magnéticas (nivel 7). Puede observarse (según se simboliza en la Figura 10.1), que cuanto más alto es el nivel menor es su capacidad, pero la velocidad es mayor; en definitiva a medida que un nivel está más próximo al procesador se implementa con una tecnología más rápida y más costosa, lo que implica una capacidad menor. En concreto, se verifica:

[10.1]
$$s_i < s_{i+1}; \quad t_i < t_{i+1}; \quad b_i > b_{i+1}; \quad c_i > c_{i+1}$$

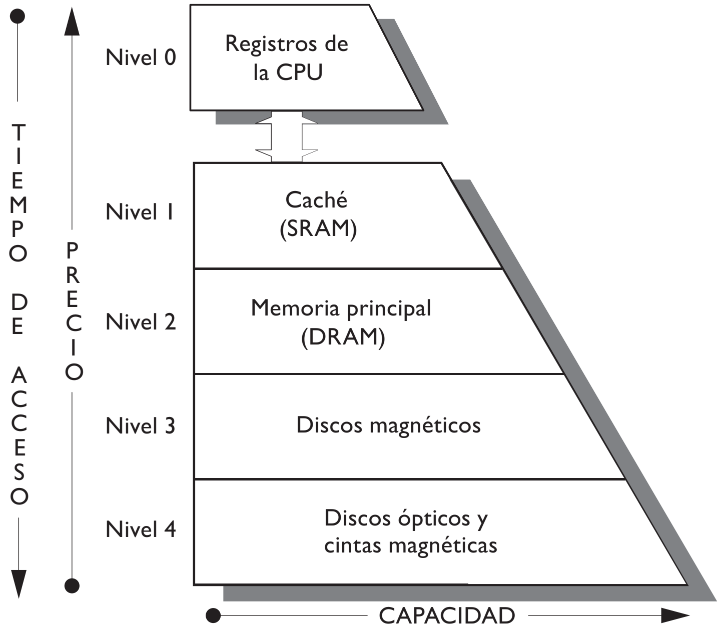


Figura 10.1. *Jerarquía de memoria.*

Los niveles superiores, concretamente el de registros, caché y memoria principal, se suelen agrupar con el nombre de **memoria interna**, y los inferiores, discos magnéticos y ópticos y cintas magnéticas, como **memoria externa**.

El procesador es el elemento principal del computador, ya que desde allí se controla el funcionamiento completo de éste y en él se hace el tratamiento de los datos. Interesa que las instrucciones y los datos con los que en un momento dado va a operar el procesador estén lo más próximos a él; es decir, en el nivel más alto de la jerarquía.

En general, cuando se solicita el contenido de una dirección de memoria en un determinado nivel y se encuentra allí se dice que se ha producido un **acierto**, y si no es así, un **fallo**. Suele satisfacerse la propiedad de **inclusión**, según la cual la información en un determinado nivel se encuentra replicada en niveles inferiores. Cuando se produce un fallo en un determinado nivel (i), se copia el bloque de datos del nivel inmediatamente inferior (i+1), dentro del cual se encuentre la información solicitada: en el caso de la memoria virtual la transferencia se produce entre disco y memoria principal y el bloque de datos se suele denominar **página**. Una vez que una dirección está en los niveles superiores de la jerarquía (registros o caché) el procesador accede directamente a leer o escribir. Existen sistemas de **escritura inmediata** en los que en las operaciones de escritura se copian los datos en todos los niveles de la jerarquía donde se encuentran. Se obtienen mejores prestaciones con los **sistemas de postescritura**, según los cuales inicialmente sólo se modifican los datos en el nivel superior. En este caso, cada bloque mantiene un **bit de modificación** (*bm*) que se hace cero en el momento de cargarse el bloque en un nivel, y se hace uno en caso de que el procesador escriba en él; en otras palabras, el bit de modificación indica si el contenido del bloque coincide ($bm = 0$) o no ($bm = 1$) con la copia de ese bloque en el nivel inmediatamente inferior.

Según se van ejecutando los programas el nivel correspondiente se irá llenando de información, llegándose a la situación en la que esté completamente lleno. En este caso al

tener que introducir un nuevo bloque, un **algoritmo de reemplazo** debe decidir qué bloque de datos debe desalojarse para acoger al nuevo. En caso de que el bit de modificación del bloque a desalojar sea cero, puede sobrescribirse sin más, ya que existe una copia exacta de él en el nivel inmediatamente inferior; si, por el contrario, el bit de modificación es uno, antes de alojar el nuevo bloque hay que actualizar la copia del antiguo existente en el nivel inmediatamente inferior.

Existen estrategias para determinar qué información en un momento dado debe ubicarse en cada uno de los niveles superiores, de forma que se produzca el menor número posible de fallos en los accesos a datos en un determinado nivel. Así, si se está ejecutando un determinado programa o utilizando un grupo de datos es muy probable que si se referencia a un elemento, los elementos cercanos a él tiendan a ser referenciados pronto (principio denominado de **localidad espacial**). También, debido a que los bucles son muy frecuentes en programación, si se referencia un elemento, tenderá a ser nuevamente referenciado pronto (principio de **localidad temporal**). Estos dos principios han inspirado distintos algoritmos de reemplazo y procedimientos para gestionar la memoria caché y la memoria virtual.

La **memoria virtual** es una técnica que resulta de aplicar los principios anteriores, y que se analizará en detalle en la Sección 13.6.5. Con ella el programa y sus datos se mantienen en disco, y sólo la parte de ellos implicada en la ejecución se lleva a la memoria interna.

Se denomina **tasa de aciertos**, $\tau_{aciertos,i}$, de un determinado nivel i al cociente entre el número de accesos realizados con éxito y el número total de accesos a ese nivel; y **tasa de fallos**, $\tau_{fallos,i}$, de un determinado nivel i al cociente entre el número de accesos realizados sin éxito y el número total de accesos a ese nivel. También se suele medir el volumen de aciertos y fallos en porcentajes (%).

Conociendo el tiempo de acceso de cada sistema de memorización en cada nivel, podemos obtener el tiempo medio de acceso a un nivel i con la siguiente expresión:

[10.2]

$$\boldsymbol{t_{a,i} = \tau_{aciertos,i} \cdot t_i + \tau_{fallos,i} \cdot t_{i+1}}$$

donde t_i representa el tiempo de acceso del sistema de almacenamiento del nivel i, y t_{i+1} el del nivel inmediato inferior dentro de la jerarquía mostrada en la Figura 10.1.

EJEMPLO 10.2

Suponiendo un sistema que no dispusiese de memoria virtual, y con tan sólo dos niveles de jerarquía (caché y memoria principal). Si los tiempos de acceso de los dispositivos utilizados son 5ns y 50 ns, respectivamente, y el porcentaje de aciertos en caché del 95%. ¿Qué porcentaje de mejora se ha obtenido en el tiempo de ciclo al haber introducido la caché?

Si no hubiese caché, el tiempo medio de acceso sería el de la memoria principal: 50 ns, ya que el procesador efectuaría todos los accesos a ella.

Con memoria caché el tiempo medio de acceso sería, de acuerdo con la expresión [10.2]:

$$t_{acceso} = \tau_{a,caché} \cdot t_{caché} + \tau_{f,caché} \cdot t_{mp} = (0{,}95 \cdot 5) + (0{,}05 \cdot 50) = 7{,}25\,ns$$

Es decir, el tiempo de acceso del sistema de caché mejora con respecto al sistema sin ella en un:

$$mejora = \frac{50 - 7{,}25}{50} \cdot 100 = 85{,}5\%$$

10.2 Memoria interna

En esta sección se considerarán las unidades de memoria interna, formadas por circuitos integrados (memoria semiconductora), y que tienen mayor relación directa con el procesador: la memoria caché (Sección 10.2.1) y la memoria principal (Sección 10.2.2). Los otros elementos de le memoria interna (registros) ya han sido considerados ampliamente en los Capítulos 6 al 9.

Existen dos tipos de memorias de semiconductor que pueden utilizarse para configurar la memoria interna del computador. Por un lado están las memorias **RAM estáticas** o **SRAM** (*Static Random Access Memory*) en las que cada **celda de memoria** (circuito capaz de almacenar un bit) está constituida por un biestable de unos 6 transistores. En cambio, una celda de una memoria **RAM dinámica** o **DRAM** (*Dynamic Random Access Memory*) consta básicamente de tan solo un transistor funcionando como condensador que almacena la carga correspondiente al valor binario que memoriza. Las memorias SRAM son más rápidas pero al ser más complejas son menos miniaturizables que las DRAM teniendo cada chip mucha menos capacidad de almacenamiento.

Existen diversos parámetros para determinar las prestaciones de la memoria; como son los siguientes:

- **Tiempo de acceso a memoria** (t_a) o **latencia**: es el tiempo que transcurre desde el instante en que se presenta una dirección a la memoria y el instante en el que el dato queda memorizado o está disponible para ser usado.
- **Tiempo de ciclo de memoria** (t_c): el tiempo mínimo que debe transcurrir entre dos accesos sucesivos.
- **Ancho de banda** (*AB*): número máximo de bytes que se pueden transmitir por segundo entre la memoria y el procesador. Este parámetro no sólo depende del tiempo de acceso a memoria, si no también del número de bytes a los que se puede acceder en paralelo y de la capacidad de transferencia del bus de interconexión entre memoria y procesador.

En las secciones siguientes (Secciones 10.2.1 y 10.2.2) analizaremos con más detalles estos dos tipos de memoria.

10.2.1 Memoria caché

Aunque la memoria principal es muy rápida (tiempos de acceso a una posición de memoria del orden de 30 a 200 ns, dependiendo de los circuitos y configuración utilizados) es del orden de 15 a 60 veces más lenta que el procesador (entre 0,25 a 0,5 ns), por lo que esta última unidad se ve frenada considerablemente cuando tienen que captar o escribir una palabra de memoria. Una forma de paliar este problema es utilizar una **memoria caché** que es un sistema de almacenamiento de tecnología más rápida intermedia entre la memoria principal y el procesador. La memoria caché suele estar constituida por circuitos integrados SRAM que son más rápidos (aproximadamente de 8 y 16 veces, con tiempos de acceso entre 0,5 y 25 ns) que los DRAM (30 a 200 ns) que forman la memoria principal; pero, como son más caros, voluminosos (unas 16 veces) y de mayor consumo energético, la capacidad de ellos que se suele incluir en un computador es de unas 1.000 a 100 veces menor. La memoria caché es usada por el sistema de memoria para mantener la información más comúnmente usada por el procesador, evitando así los relativamente lentos accesos a la memoria principal. Los microprocesadores modernos incluyen en su propio chip total o parcialmente su caché.

EJEMPLO 10.3

El Intel Pentium 4 dispone de una memoria caché interna (dentro del propio chip) de dos niveles (L1 y L2). El nivel L1 (más rápido pero menos capaz que el L2) está, a su vez, dividido en dos bloques: uno (L1I) de 96 KBytes para microoperaciones y otro (L1D) de 8 KB para datos. El otro nivel, L2, es tanto para instrucciones como para datos y puede ser de 256 ó 512 KBytes. El procesador accede directamente al nivel 1 (L1I y L1D), y el nivel 1 se comunica con el exterior tan sólo a través del nivel 2 (L2). Otros procesadores, como los primeros Pentium o el USPARC III, están diseñados de forma que la caché de nivel 2 (L2) debe instalarse externamente (hasta 8 MBytes en el caso del UltraSPARC III).

Tanto la memoria caché como la memoria virtual (que se analizará en la Sección 13.6.5) se fundamentan en el concepto de **localidad de las referencias**, que se basa en los principios de localidad espacial y temporal, citados anteriormente (Sección 10.1). Como consecuencia, cuando el procesador requiere una información determinada, son tácticas muy adecuadas:

a) Recuperar no sólo el dato o instrucción requerida sino también los de direcciones próximas a él (consideración espacial). Ese grupo de direcciones adyacentes se denomina **bloque de datos**.
b) Almacenar temporalmente el bloque de datos recuperado, en un subsistema con tiempo de acceso lo más próximo posible al del procesador, ya que con gran probabilidad en un plazo de tiempo pequeño volverá a ser solicitado (consideración temporal).

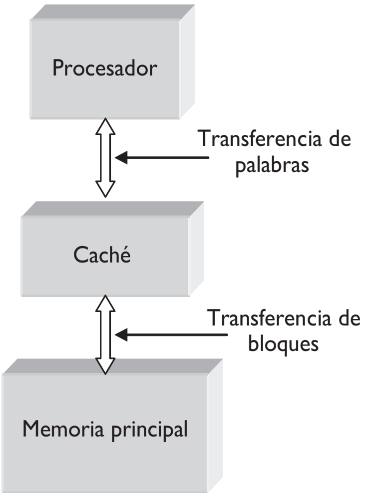


Figura 10.2. *Conexión de la memoria caché.*

La memoria caché es una memoria que se sitúa entre el procesador y la memoria principal (Figura 10.2) y, simplificadamente, funciona de la manera que se describe a continuación. El procesador genera peticiones de acceso a memoria (lectura o escritura) proporcionando direcciones de posiciones de memoria. El controlador de la caché, a partir de la dirección emitida por el procesador comprueba si su contenido está o no en la caché; si lo está, se produce un

acierto en la caché y se realiza la lectura o escritura en la caché. Si, por el contrario, no lo está, se produce un **fallo en la caché**, y como consecuencia de ello se carga en la caché un bloque de datos (**línea de caché)**, que es un conjunto de palabras (64 bytes, por ejemplo) con direcciones consecutivas que contiene la instrucción o dato solicitado y, simultáneamente, se pasa el dato o instrucción solicitado al procesador. Una vez que un bloque de datos se lleva a la caché, se deja allí el mayor tiempo posible. Claramente el concepto de memoria caché aprovecha los principios de localidad espacial (se almacenan en ella además de los datos solicitados los próximos) y de localidad temporal (los bloques de datos se mantienen en lo posible en la caché).

10.2.2 Memoria principal

En la Sección 7.1 se describió el funcionamiento de la memoria principal. En resumen, la memoria principal de un computador está organizada en grupos de celdas de memoria denominados **palabras de memoria**. Una palabra es el conjunto de bits que se puede leer o memorizar en un instante dado, y su número de bits, n, se denomina **ancho** o **longitud de palabra** o **ancho de la memoria**. La información, ya sean instrucciones o datos, se almacena en palabras de memoria, y ésta la podemos imaginar como un conjunto ordenado de palabras, a cada una de las cuales se puede acceder, para recuperar o memorizar, indicando su **dirección** (posición relativa). Las longitudes usuales de palabras de memoria son 8, 16, 32 ó 64 bits. Para acceder (leer o escribir) a las posiciones de memoria, ésta dispone de un **bus de direcciones** de m hilos y uno o dos **buses de datos** de n bits, para entrada o salida de la información (palabras) grabada en ella. La capacidad máxima, en bytes, de la memoria de un sistema viene dada por:

[10.3]

$$C_{MP\,\max} = \frac{n \cdot 2^m}{8}\ Bytes$$

A continuación se describen brevemente: los tipos de circuitos integrados que suelen utilizarse para construir la memoria principal (circuitos DRAM, Sección 10.2.2.1), la forma de ensamblarlos en módulos de memoria para poderlos conectar adecuadamente en el computador (Sección 10.2.2.2) y, por último, la distintas formas de acceder a los datos (Sección 10.2.2.3).

10.2.2.1 Tecnologías de circuitos RAM dinámicos

Como se comentó al inicio de esta Sección 10.2, en una memoria RAM dinámica cada bit se almacena en forma de carga eléctrica en una celda constituida por un diminuto condensador. Esta carga se pierde o deteriora según transcurre el tiempo y en los procesos de lectura, por lo que los chips de este tipo de memoria contienen internamente circuitos dedicados a *refrescar* la información periódicamente (cientos de veces por segundo) o antes de realizar un acceso a la misma (precarga). Obviamente, en los intervalos de tiempo en los que se realiza la operación de refresco no se puede realizar ningún acceso, limitando la velocidad de respuesta. Las memorias SRAM RAM no presentan este problema al estar formadas por celdas en las que cada una de ellas existe un biestable (Sección 6.6), lográndose, como se ha comentado anteriormente, velocidades mayores.

Un chip de memoria se puede considerar formado por una matriz o **núcleo de celdas de memoria**, donde cada una de ellas almacena un bit, y una circuitería auxiliar que sirve

de: interfaz con el exterior del chip, para refresco (caso de DRAM), y para seleccionar, a partir de la dirección de entrada, la posición a la que hay que acceder para leer o escribir una palabra. Para seleccionar una posición a partir de una dirección de entrada, disponen de un decodificador (Sección 5.4.5 y 6.6). Así, un chip con un espacio de direcciones de 1 Mpalabras, necesita un decodificador de 20 entradas a 1.048.576 salidas. La complejidad de este decodificador es muy grande y su implementación ocupa una gran cantidad de espacio dentro del chip que podría utilizarse para incrementar el tamaño del núcleo de celdas, y por tanto su capacidad de almacenamiento. Este tipo de arquitectura se denomina **configuración de memoria 2D** (dos dimensiones). Es mejor utilizar una **configuración de memoria 3D** (tres dimensiones) de forma que las posiciones se distribuyesen regularmente a lo largo de un plano de dos dimensiones (*X,Y*), y las distintas celdas o bits de una palabra determinada se encontrasen en distintos planos pero en la misma posición (*X,Y*). Para seleccionar una palabra determinada hay que proporcionar las direcciones de fila (*X*) y de columna (*Y*), de esta manera en lugar del gran decodificador indicado en el ejemplo anterior serían necesarios dos de 10 entradas a 1.024 salidas que juntos ocupan mucho menos espacio que el de una arquitectura 2D y por tanto en dicho chip se podría integrar un número mayor de celdas de memoria.

EJEMPLO 10.4

El núcleo de celdas de una memoria de 256 MB organizada en palabras de 8 bits, podría organizarse de las siguientes formas:

- *Configuración de memoria 2D: las celdas de memoria se distribuirían en un plano de* $2^8 \cdot 2^{20} = 2^{28}$ = *268$_1$435.460 filas y 8 columnas (cada una para un bit de la palabra de datos). Los circuitos auxiliares deberían contener un decodificador de 28 entradas en 268$_1$435.460 salidas.*
- *Configuración de memoria 3D: las celdas de memoria se distribuirían en 8 planos, uno por cada bit de la palabra; así, el plano inferior podría contener todos los bits de orden 0 de todas las palabras, el plano inmediato superior los bits de orden 1, y así sucesivamente. Cada plano contendría 13.384 filas y 16.384 columnas, de forma que así se podrían direccionar las* 13.384^2 = *268$_1$435.460 posiciones del núcleo de la memoria.*

Ocupan mucho menos espacio dos decodificadores de 14 entradas en 16.384 salidas, que uno de 28 entradas en 268$_1$435.460 salidas.

Existen distintas arquitecturas de circuitos integrados ideadas para conseguir un mayor grado de integración (mayor capacidad) y una mayor velocidad. El núcleo de celdas de memoria no cambia, sino los circuitos de interfaz con el exterior. La lectura o escritura de una palabra se realiza en distintas fases como pueden ser: acceso a una fila de celdas, acceso a una columna de celdas, lectura/escritura de palabra de datos y precarga de la matriz de celdas. Se logran incrementos de velocidad y disminución de los tiempos de acceso, de las siguientes formas:

- Incluyendo dentro del circuito elementos que permitan solapar las distintas fases de lectura o escritura para operaciones sucesivas. Este modo de funcionamiento se conoce con el nombre de **segmentación de cauce** o *pipe-line* (véase Sección 12.7). De esta forma, por ejemplo, se puede iniciar un acceso antes de que el anterior haya concluido (como ocurre, por ejemplo, en las **memorias EDO RAM**).

- Debido a los principios de localidad espacial y temporal muchas veces se requiere leer o escribir datos de posiciones consecutivas de memoria (**ráfagas de datos**) por lo que se incluyen sistemas que permiten que una vez se haya accedido a la primera posición las lecturas sucesivas dentro de la ráfaga sean más veloces. En consecuencia, los accesos resultan mucho más rápidos cuando se hacen en posiciones consecutivas de memoria, debiendo considerarse dos tiempos: tiempo de acceso inicial, y tiempo de transmisión de la ráfaga, alcanzándose así tiempos de acceso medio menores cuanto mayor sea la ráfaga. Este tipo de acceso por ráfagas se pueden realizar, por ejemplo, manteniendo constante la dirección de fila y leyendo consecutivamente los datos de varias columnas (**memorias FPM DRAM**) o incluyendo un contador interno al chip que vaya generando las direcciones sucesivas sin necesidad de darlas externamente (**memorias BEDO DRAM**).
- Monitorizando de forma síncrona con el reloj del procesador el funcionamiento de los circuitos de la memoria; así el direccionamiento y la transferencia de datos se efectúan de forma sincronizada con un mismo reloj permitiendo transferir ráfagas de datos a velocidades de 100 MHz a 2,1 GB/s. Es posible transmitir un dato por cada pulso de reloj (**memorias SDRAM**), dos datos (uno en el flanco de subida del pulso de reloj y otro en el de bajada, **memorias DDR SDRAM**) o incluso cuatro datos (**memorias QBM DDR SDRAM**).
- Incluir en el interior dos o más núcleos o bloques de celdas de forma que actúen en paralelo o de forma que mientras en unos se hacen el acceso en otros se realiza la precarga, etc.
- Intercambiar los datos en buses serie o de un ancho relativamente pequeño (8 a 28 bits). Las direcciones y los datos se transmiten en el mismo bus por medio de paquetes que se multiplexan en el tiempo: primero se transmiten los paquetes con direcciones y luego se intercambian los de datos. Los buses son muy cortos y optimizados (como máximo 12 cm de longitud) para conseguir grandes velocidades. Esta técnica se utiliza, por ejemplo, en las **memorias Rambus DRAM**.
- Subdividir el núcleo de celdas de memoria en pequeños bancos para reducir los tiempos de acceso (**memorias FCRAM** y **memorias MoSys**).

Tabla 10.2. *Algunas características de distintas tecnologías de chips de memoria (Fuente: TechEncyclopedia, TechWeb).*

Tipo	Año de comercialización	Frecuencia de reloj	Ancho de bus	Ancho de banda de pico	Tensión de alimentación
FPM (60, 70 ns)	1990	25 MHz	64 bits	200 MB/s	5 V
EDO (50, 60, 70 ns)	1994	40 MHz	64 bits	320 MB/s	5 V
SDRAM (66 MHz)	1996	66 MHz	64 bits	528 MB/s	3,3 V
SDRAM (100 MHz)	1998	100 MHz	64 bits	800 MB/s	3,3 V
SDRAM (133 MHz)	1999	133 MHz	64 bits	1,1 GB/s	3,3 V
RDRAM (Direct Rambus)	1999	400 MHz (x2)	64 bits	1,6 GB/s	2,5 V
DDR SDRAM (100 MHz)	2000	100 MHz (x2)	64 bits	1,6 GB/s	3,3 V
DDR SDRAM (133 MHz)	2000	133 MHz (x2)	64 bits	2,1 GB/s	3,3 V

EJEMPLO 10.5

Los chips de memorias DRDRAM **(Direct Rambus DRAM)** *están divididos en 16 bancos de celdas, y utilizan un bus que funciona a 400 MHz con 3 canales de un byte cada uno (uno para direcciones/control y dos para datos. Los bits de datos y control se transfieren en paquetes de 8 bytes consumiendo 4 ciclos de reloj. En consecuencia, el tiempo de ciclo del bus será:*

$$T_{bus} = \frac{1}{400 \cdot 10^6 \, ciclos/s} = 2{,}5 \cdot 10^{-6} s = 2{,}5 \; ns$$

Para transferir un paquete de datos de 16 bits se necesitarán:

$$t_{paquete} = 4 \, ciclos \cdot 2{,}5 \, ns/ciclo = 10 \, ns$$

Ahora bien, en cada ciclo del bus se transmiten datos tanto en el flanco de subida como de bajada de la señal de reloj, con lo que en cada 4 ciclos se transfieren 2·8 = 16 Bytes. Por consiguiente la velocidad de transferencia alcanzable será:

$$v_{transferencia} = \frac{16 \, Bytes}{10 \, ns} = 1{,}6 \, GB/s$$

Como consecuencia general de las técnicas anteriores, cuando se transfiere un bloque de datos se obtiene un rendimiento mayor que si se transfiriesen dichos datos uno a uno. La primera palabra transferida, consume un tiempo igual al de una palabra aislada (por ejemplo, 5 ciclos de reloj), pero cada una de las siguientes palabras del bloque puede consumir tan sólo un ciclo de reloj. Una vez que se dispone a la salida de la memoria de un dato leído, se consumirá al menos otro ciclo de reloj para transferir el dato por el bus al procesador, aunque esta operación de transferencia puede solaparse con un acceso posterior a la memoria.

EJEMPLO 10.6

Suponiendo un procesador que consume 6 ciclos de reloj para acceder a una palabra, y 3 ciclos para accesos sucesivos, calcular el número de ciclos que se invierten desde que el procesador solicita un bloque de 16 palabras consecutivas hasta que las recibe.

SOLUCIÓN:

Tenemos que considerar los siguientes tiempos:

- *Transmisión de la dirección procesador → memoria: 1 ciclo.*
- *Acceso a la 1ª palabra del bloque: 6 ciclos.*
- *Acceso a la 2ª palabra del bloque: 3 ciclos (en el primero de ellos simultáneamente se transmite la 1ª palabra de memoria → procesador).*
- *Acceso a la tercera palabra del bloque: 3 ciclos (en el primero de ellos simultáneamente se transmite la 2ª palabra de memoria → procesador).*
- *……*
- *Acceso a la 16ª palabra del bloque: 3 ciclos (en el primero de ellos simultáneamente se transmite la 15ª palabra de memoria → procesador).*
- *Transmisión de la 16ª palabra de memoria → procesador: 1 ciclo.*

Es decir:

[10.4]

$$N_c = 1 + 6 + 16 \cdot 3 + 1 = 56 \; ciclos$$

En la Tabla 10.2 se incluye un resumen comparativo de distintos tipos de tecnologías de memorias RAM dinámicas. Más detalles pueden encontrase en las referencias [Jul05, Sar02].

10.2.2.2 Módulos de memoria principal

Comercialmente las memorias se suministran en pequeñas tarjetas de circuito impreso denominados **módulos de memoria** que agrupan varios circuitos integrados y que se insertan directamente en las regletas de conectores apropiadas de la tarjeta base donde se encuentra el microprocesador (Sección 12.6). Las tres agrupaciones más conocidas son:

- **SIMM** (*Single In-line memory Module*), que pueden contener 8 chips de 32 o de 64 Mbits cada uno totalizando un módulo con una capacidad total de 32 ó 64 Mbytes, respectivamente. Hay versiones con conectores de 30 ó 72 contactos, según sea de 8 ó 32 bits el ancho de palabra (Figura 10.3).
- **DIMM** (*Dual In-line Memory Module*), pueden almacenar 64 ó 128 MB e incluso capacidades superiores. Los contactos están por las dos superficies de la tarjeta, teniendo las primeras versiones 84 por cada lado (168 en total, Figura 10.3). En un instante dado es capaz de leer o escribir datos de 64 bits (ancho del bus de datos de 64 hilos). Estos módulos se utilizan con los primeros chips SDRAM, existiendo en la actualidad módulos DIMM de 184 y 240 contactos para las memorias SDRAM de tipo DDR y DDR2, respectivamente.
- **SODIMM** (*Small Outline-DIMM*), son módulos miniaturizados que realizan funciones análogas a los anteriores y que han sido ideados para portátiles y pequeños equipos.
- **RIMM** (*Rambus In line Memory Module*), son como los DIMM, con 184 pines, pero tienen una asignación de conectores distinta, y se usan como módulos de las memorias Direct RDRAM. Además incluyen un radiador metálico para disipar mejor el calor.

En la Tabla 10.3 se incluye una lista de los tipos de módulos más utilizados.

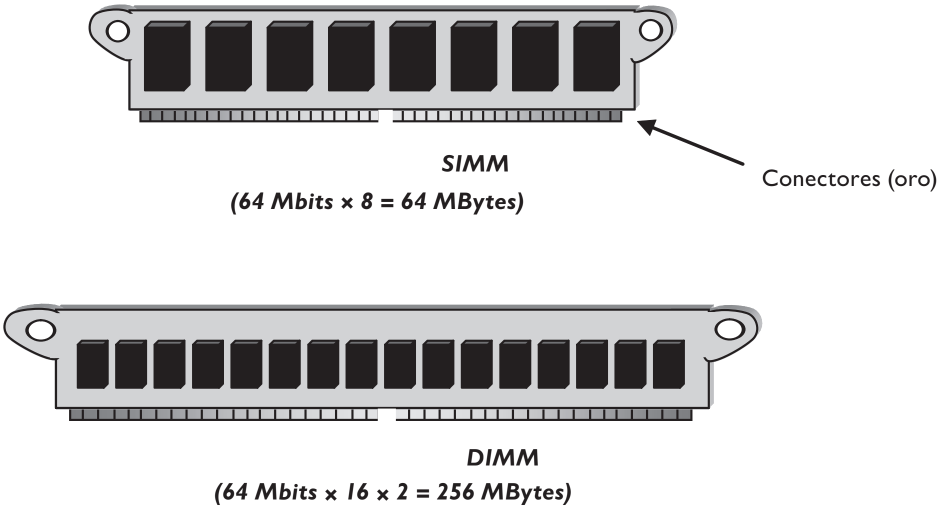


Figura 10.3. *Esquema simplificado de un módulo de memoria: (a) SIMM de 64MB, y (b) DIMM de 256MB.*

Tabla 10.3. *Tipos de módulos de memoria.*

Tipo	Nº de pines	Tecnologías de chips que incluyen
SIM	30	DRAM
	72	FPM
DIMM	168	FPM, EDO, SRAM
	184	DDR, SRAM
	240	SRAM DDR2
SODIMM	72	FPM, EDO
	144	FPM, EDO, SDRAM
	200	DDR, SDRAM
RIMM	184	RDRAM (Rambus)

10.2.2.3 Acceso a los datos

Según indicamos anteriormente, una palabra de memoria es la información que puede leerse o escribirse en un instante de tiempo. En cuanto a las unidades de información direccionables pueden considerarse dos tipos de memoria:

- **Memorias direccionables por palabras**, en las que cada dirección corresponde a una palabra de memoria, de forma que dos direcciones consecutivas corresponden a dos palabras consecutivas de memoria.
- **Memorias direccionables por bytes**, en las que las direcciones corresponden a bytes de memoria. Ésta es la forma usual de direccionamiento que usan, por ejemplo, los PC compatibles. En este caso las direcciones de inicio de cada palabra son múltiplos del número de bytes que incluye cada palabra, según se pone de manifiesto en el siguiente ejemplo.

EJEMPLO 10.7

Sea una memoria direccionable por bytes y de longitud de palabra de $n = 32$ bits. En esta memoria, por lo tanto, en cada palabra caben 4 bytes (Figura 10.4), y sus direcciones de comienzo serán: 0, 4, 8, 16, etc… Obsérvese que, en binario, todas las direcciones de palabra acabarán con dos ceros, de forma de que si $m = 32$, las direcciones de las palabras serían las que se indican en la Figuras 10.4 y 10.5.

Puede concluirse de este ejemplo que de los 32 bits de dirección, los 30 más significativos identifican la palabra, y los dos menos significativos el byte dentro de la palabra. Si el ancho de la memoria fuese de 64 bits, de los 32 bits de dirección, los 29 más significativos identificarían la palabra, y los tres menos significativos el byte dentro de la palabra; es decir, las direcciones de los bytes de inicio de las palabras acabarían siempre con tres ceros.

Cuando las palabras se organizan en la forma anteriormente indicada se tienen **direcciones alineadas**; ahora bien, como en realidad el direccionamiento de la memoria se hace por

bytes, una palabra puede comenzar en cualquier posición arbitraria, y si no coincide con un múltiplo del número de bytes por palabra se tienen **direcciones no alineadas**. Si la memoria se direcciona por bytes y se quieren obtener direcciones alineadas los dos o tres bits menos significativos tienen que ser 0, dependiendo de que el ancho de la memoria fuese de 32 ó 64 bits, respectivamente.

Palabra 0	Byte 0000 0000	Byte 0000 0001	Byte 0000 0002	Byte 0000 0003
Palabra 1	Byte 0000 0004	Byte 0000 0005	Byte 0000 0006	Byte 0000 0007
Palabra 2	Byte 0000 0008	Byte 0000 0009	Byte 0000 000A	Byte 0000 000B
Palabra 3	Byte 0000 000C	Byte 0000 000D	Byte 0000 000E	Byte 0000 000F
Palabra 4	Byte 0000 0010	Byte 0000 0011	Byte 0000 0012	Byte 0000 0013
Palabra $2^{30}-1$	Byte FFFF FFFC	Byte FFFF FFFD	FFFF FFFE	FFFF FFFF

Figura 10.4. *Distribución de palabras y bytes en una memoria de palabras de n = 32 bits.*

Nº de palabra	Dirección en binario del inicio de cada palabra (bytes)	Dirección HEX
0	0000 0000 0000 0000 0000 0000 0000 0000	0000 0000
1	0000 0000 0000 0000 0000 0000 0000 0100	0000 0004
2	0000 0000 0000 0000 0000 0000 0000 1000	0000 0008
3	0000 0000 0000 0000 0000 0000 0000 1100	0000 000C
4	0000 0000 0000 0000 0000 0000 0001 0000	0000 0010
5	0000 0000 0000 0000 0000 0000 0001 0100	0000 0014
6	0000 0000 0000 0000 0000 0000 0001 1000	0000 0018
7	0000 0000 0000 0000 0000 0000 0001 1100	0000 001C
8	0000 0000 0000 0000 0000 0000 0010 0000	0000 0020
...	...	...
$2^{30}-4$	1111 1111 1111 1111 1111 1111 1111 0000	FFFF FFF0
$2^{30}-3$	1111 1111 1111 1111 1111 1111 1111 0100	FFFF FFF4
$2^{30}-2$	1111 1111 1111 1111 1111 1111 1111 1000	FFFF FFF8
$2^{30}-1$	1111 1111 1111 1111 1111 1111 1111 1100	FFFF FFFC

Figura 10.5. *Direccionamiento por bytes en una memoria con n = 32 bits.*

Cuando una palabra incluye diversos bytes de un mismo dato o instrucción, según se comentó en la Sección 4.5 (Figura 4.6) el byte almacenado en la parte más baja de las direcciones de memoria puede albergar los 8 bits menos significativos del dato o instrucción, o los más significativos, según se siga el criterio del extremo menor o el criterio del extremo mayor, respectivamente.

En general, los accesos a memoria pueden realizarse de tres formas:

- **Acceso por palabras**: ésta es la forma normal de acceso.
- **Acceso por bytes**: la memoria lee todos los bytes que componen la palabra a la que pertenece el byte, pero los no solicitados son ignorados por el procesador. En el caso de escritura de un byte, la memoria dispone de circuitos de control que únicamente hacen escribir el byte implicado en la operación, dejando inalterados los otros bytes de la palabra a la que se accede.
- **Acceso por bloques (ráfagas)**: con una gran frecuencia se solicita la lectura o escritura de posiciones consecutivas de memoria; tal es el caso que se presenta cuando se carga un programa en la memoria principal, o se salva el contenido de un archivo en memoria secundaria. Es posible transferir bloques de información, siendo necesario indicar al sistema de memoria tan sólo la dirección inicial del bloque y su tamaño.

Existen diversas técnicas para mejorar la velocidad de acceso a memoria; una de ellas se denomina **entrelazado de memoria**. Consiste en disponer varios, *p*, módulos de memoria en paralelo. Las conexiones se realizan de forma tal que los distintos módulos guardan posiciones consecutivas de memoria (Figura 10.6); así el módulo *0* contendrá las posiciones *0, p, 2p, 3p, …*; el segundo módulo *1, p + 1, 2p + 1, 3p + 1,* etc.; y el módulo *p – 1*, las posiciones, *p – 1, 2 · (p – 1)*, etc. Cuando se desea acceder a una determinada palabra, en realidad se acceden a *p* palabras consecutivas. El esquema de direccionamiento se muestra en la Figura 10.6; si cada módulo almacena $q = 2^j$ palabras y suponiendo que el número de módulos es $p = 2^k$, los primeros *j* bits seleccionarán a la palabra de dirección *j* dentro de cada módulo, y los últimos *k* bits al módulo que contiene la palabra solicitada.

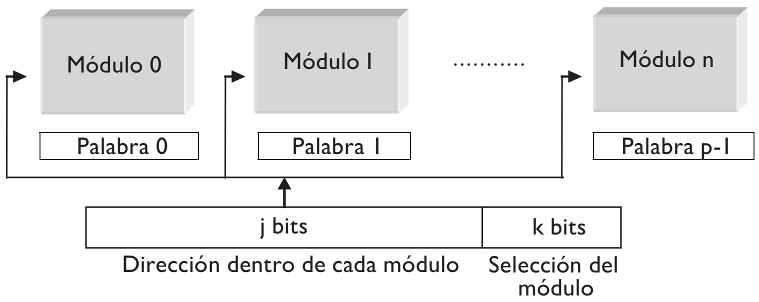


Figura 10.6. *Esquema simplificado de un sistema de memoria entrelazada con cuatro módulos.*

EJEMPLO 10.8

Una memoria dispone de 16 módulos entrelazadazos, cada uno de ellos de 1Mpalabra. El esquema de direccionamiento de memoria sería el siguiente:

- *Para direccionar dentro de cada módulo, serán necesarios 20 bits, ya que* 2^{20}*= 1Mega.*
- *Como hay 16 módulos, serían necesarios 3 bits para identificar cada uno de ellos, ya que* $2^4 = 16$*.*

En consecuencia, las direcciones serían de 24 bits. Al leer, por ejemplo, la dirección A35B78, se leería en los 16 módulos 16 las palabras de posición A35B7, y de todas ellas la palabra solicitada sería la del módulo 8.

10.3 Memoria externa

En este apartado vamos a presentar una serie de dispositivos, basados en principios magnéticos y ópticos, que son periféricos que actúan como prolongación de la memoria principal. El conjunto se estos dispositivos constituye la **memoria externa** del computador, y trata de solventar el problema de la volatilidad y de la relativa pequeña capacidad de la memoria interna. Los principales soportes que se utilizan como memoria externa, y que describiremos en las secciones siguientes, son:

- **Memorias magnéticas:**
  - Disco magnético
  - Cinta magnética
- **Memorias ópticas:**
  - CD-ROM (Disco compacto)
  - DVD-ROM (Disco digital versátil)
- **Memorias magneto-ópticas.**
- **Memorias flash USB.**

En esta sección, en primer lugar (Sección 10.3.1) daremos unas ideas básicas comunes a todos los dispositivos magnéticos, indicando la forma de grabar y de leer la información. A continuación se estudiarán los discos (Sección 10.3.2) y cintas magnéticos (Sección 10.3.3). Posteriormente se analizarán los discos ópticos (Sección 10.3.4), y los discos magneto-ópticos (Sección 10.3.5). Por último (Sección 10.3.6), describiremos brevemente las memorias flash USB.

10.3.1 Escritura y lectura de información en forma magnética

Los discos y cintas magnéticas contienen soportes de información constituidos por un sustrato, de plástico o aluminio, recubierto por un material magnetizable, tradicionalmente óxido férrico u óxido de cromo (Figura 10.7). La información se graba en unidades elementales o **celdas** que forman líneas o **pistas**. Cada celda puede estar sin magnetizar o estar magnetizada en uno de dos estados o campos magnéticos: norte (N) o sur (S). Los ceros y unos se pueden representar de muy diversas formas: magnetización N o S, cambio o no cambio de magnetización respecto a la celda anterior, etc. La celda se comporta como un elemento de memoria ya que almacena un bit. Para escribir o leer en una celda se utilizan señales eléctricas que actúan en una **cabeza** o **cápsula de lectura/escritura**, como la que esquemáticamente se muestra en la Figura 10.7.

Para escribir en una celda, una vez posicionada la cabeza sobre ella, se hace pasar por el devanado de escritura un pulso de corriente (del orden de 10 a 200 mA), que crea un campo magnético dentro del cual queda inmersa la celda del soporte. Dependiendo del sentido de la corriente (como el que se indica en la figura, o en sentido contrario) así será la polaridad del campo magnético creado y, por tanto, el estado en que queda magnetizada la celda.

La información de una celda se lee por medio de otro devanado y un amplificador-sensor. Como es conocido, un flujo magnético variable *induce* sobre una espira o bobina una fuerza electromotriz (tensión). Ésta tendrá una polaridad (+ o –, respecto de tierra) impuesta por el estado de magnetización (S o N) y el sentido de arrollamiento de la bobina. En la lectura se obtienen tensiones del orden de 0,1 a 100 mV. Las bobinas son usualmente de 10 a 1.000 espiras, y sus inductancias aproximadamente están comprendidas entre 10 μH y 100 mH. La separación de las bobinas con respecto a la superficie de la pista es de 1 a 600 μm.

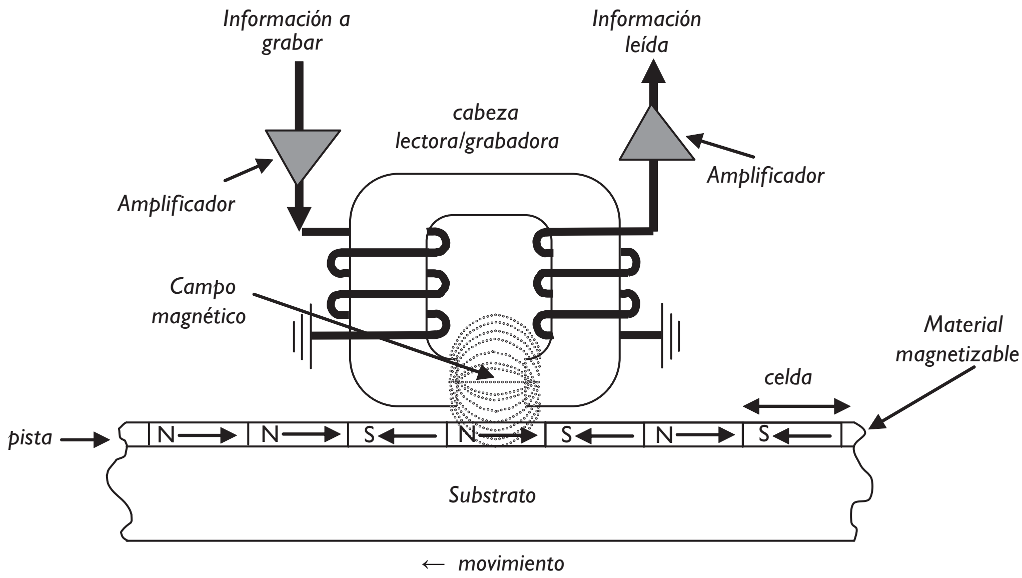


Figura 10.7. *Esquema que muestra el fundamento de la grabación y lectura en un soporte magnético por una cabeza lectora/grabadora.*

En 1990 IBM desarrolló otro tipo de cápsulas lectoras, denominadas **cabezas magnetorresistivas**. Se fundamentan en el uso de materiales cuya resistencia eléctrica varía con el campo magnético en el que se encuentren inmersos. Se aplica al dispositivo magnetorresistivo una corriente eléctrica constante, y se obtiene una tensión eléctrica cuyo valor depende del campo magnético detectado. De esta forma se puede leer la información de la pista del disco (la escritura se sigue haciendo por inducción).

También IBM, en 1991, propuso cambiar el recubrimiento de óxido magnetizable del sustrato por una **película delgada** (microscópica) de metal puro que se puede realizar por deposición al vacío u otros métodos utilizados en el proceso de fabricación de circuitos integrados. Ese sistema tiene tres ventajas importantes: (1) al ser el grosor de la capa magnetizable menor, se pueden conseguir densidades de grabación mayores, (2) los campos magnéticos producidos son más inmunes a los cambios (el medio es más **coercitivo**), y (3) la capa magnetizable queda fuertemente unida al sustrato dando mayor consistencia al conjunto.

A diferencia de las memorias de semiconductor, la información grabada en un soporte magnético permanece sin necesidad de ningún aporte constante de energía.

En la Figura 10.8 se muestran los códigos que se suelen utilizar para grabar información digital en una superficie magnética. Dependiendo del tipo de soporte y del modelo de dispositivo se utiliza uno u otro. A continuación se describen brevemente estos códigos.

Vuelta a cero (RZ, *return to zero***)**

El 1 lógico se graba magnetizando el centro de la celda. El 0 es ausencia de magnetización.

Vuelta a saturación (RS, *return to saturation***)**

La información se representa así: el 1 por un estado de magnetización en el centro de la celda. El resto de la celda, así como el 0 lógico, se representa con el estado de magnetización opuesto.

Retorno a cero bipolar (BRZ, *bipolar return to zero***)**
El 1 se representa por un estado de magnetización en el centro de la celda, y el 0 por el estado de magnetización opuesto, también en el centro de la celda, los extremos de la celda quedan sin magnetizar.

Sin retorno a cero (NRZ, *non-return zero***)**
El 1 se representa por un estado de magnetización en toda la celda, y el 0 por el estado contrario también en toda la celda.

Sin retorno a cero inverso (NRZI, *non-return to zero invert***)**
El 1 lógico se representa por cambio de magnetización en toda la celda, sobre el estado de la celda anterior. El 0 se representa por ausencia de cambio de flujo.

Codificación en fase (PE*, phase encoding***)**
El 1 se representa por un cambio del sentido del flujo magnético (de sur a norte, por ejemplo), y el cero por el cambio contrario (norte a sur). Estos cambios se realizan en el centro de la celda. Para poderse detectar dos ceros o dos unos consecutivos, al comienzo del segundo se efectúa también un cambio.

Modulación de frecuencia (FM, *frequency modulation)*
Siempre se produce un cambio al comienzo de la celda, en el centro se producirá un nuevo cambio de magnetización si y sólo si se graba un 1.

Coeficiente de pulso (PR, *pulse ratio)*
La celda tiene un primer intervalo de longitud R1 en un estado de magnetización (N, por ejemplo), y el segundo intervalo, de longitud R2, en el estado contrario (S, en el ejemplo). El 1 lógico se representa por un pulso tal que R1/R2<1, y el 0 lógico por un pulso tal que R1/R2>1.

FM modificada (MFM*, modifed FM)*
Similar a FM excepto que el cambio de estado al inicio de la celda se efectúa sólo si el bit precedente y actual son ambos 0.

Modulación de frecuencia modificada-modificada (M²FM, *modified-modified FM***)**
Igual que FM pero sólo se efectúa el cambio de estado al comienzo de la celda si la celda anterior no contiene ningún cambio (ni al principio ni en el centro), y la celda actual no es un 1.

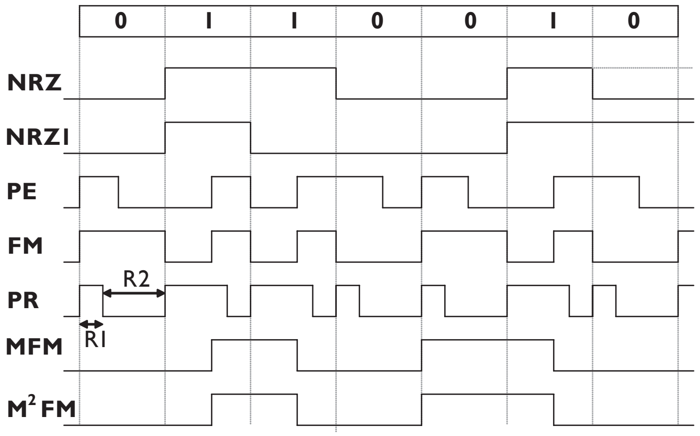


Figura 10.8. *Códigos utilizables para la grabación de información digital en un soporte magnético.*

A lo largo del estudio de las unidades de memoria auxiliar nos referiremos a estos tipos de grabación, indicando cuándo se utilizan unos u otros. Para grabar con fiabilidad la información es frecuente efectuar una transformación de código previa (además de introducir bits de paridad). Así, por ejemplo, el código NRZI modificado (**MNRZI**) transforma grupos de 4 bits en grupos de 5 bits. El código nuevo es tal que nunca se graban más de dos ceros seguidos (véase Ejercicio 10.10).

Los dispositivos más modernos utilizan el sistema de **grabación RLL 3,9** (**RLL avanzada**). Hace corresponder a cada byte un patrón predeterminado de grabación de 16 bits, de forma tal que siempre hay entre cada dos unos consecutivos (transiciones), de 3 a 9 ceros (el mínimo espaciamiento entre inversiones es 4 y el máximo 10). Así se consiguen densidades de grabación del doble de las técnicas usuales en **doble densidad de grabación**.

Más detalles sobre los códigos pueden verse en la referencia [Mig04].

La información contenida en un soporte magnético se transfiere desde y hacia el procesador o memoria principal no celda a celda, sino a ráfagas de información, denominadas **bloques** o **registros físicos**. El tiempo que se tarda en acceder a un registro determinado es variable, dependiendo, entre otras cosas, del lugar previo donde se encuentra la cabeza lectora. Se denomina **tiempo de acceso (medio)** al tiempo que por término medio se tarda en acceder a cualquier registro físico. Si para acceder a un bloque concreto es necesario que la cabeza lectora vaya recorriendo (o leyendo) uno a uno los bloques que hay desde su posición inicial a la final, se dice que el dispositivo es de **acceso secuencial**. Si, por el contrario, la cabeza lectora puede posicionarse directamente en un registro dado (indicando su posición física) se dice que el dispositivo es de **acceso directo**. Los circuitos de la memoria central, los tambores, y los discos magnéticos son unidades de acceso directo, por el contrario las cintas magnéticas son de acceso secuencial. Los dispositivos de acceso directo son mucho más rápidos que los de acceso secuencial.

10.3.2 Discos magnéticos

Los discos magnéticos son sistemas de almacenamiento de información que en la actualidad constituyen el principal soporte utilizado como memoria auxiliar, tanto en los microcomputadores como en grandes sistemas informáticos. A pesar de que son más costosos que las cintas magnéticas y discos ópticos, tienen la ventaja sobre éstos de que son sistemas de acceso directo, con lo que se consiguen tiempos medios de acceso, del orden de 10 a 100 milisegundos, sustancialmente menores que con los discos ópticos y las cintas magnéticas.

10.3.2.1 Principios de funcionamiento

Los distintos tipos de discos magnéticos se fundamentan en la grabación magnética de la información en las superficies de un plato o disco circular recubierto de la capa magnetizable (óxido o película delgada). El plato o disco puede ser de plástico flexible, o puede ser rígido (usualmente de aluminio). En el primer caso tenemos disquetes (discos flexibles) y en el segundo caso discos rígidos o *duros* (*hard disks*).

Tanto en los discos duros como flexibles la información se graba en circunferencias concéntricas. Cada una de las circunferencias concéntricas grabadas constituye una **pista** (Figura 10.9), que se consideran numeradas correlativamente de fuera a dentro, empezando por cero. Así mismo el disco se considera dividido en arcos iguales denominados **sectores**, de forma que cada pista está compuesta por sectores. Los sectores también se consideran

numerados en una secuencia única para todo el disco, y la capacidad de información del usuario que suele almacenarse en un sector es de 512 Bytes.

La unidad física de lectura/escritura es el sector, y ésta es la unidad utilizada, por ejemplo, por el sistema de archivos NTFS. No obstante, hay sistemas operativos, que utilizan como unidades de transferencia conjuntos de un número determinado de sectores, que denominan **unidades de asignación** (*clusters*). El número de sectores que conforma una unidad de asignación depende del tamaño y tipo de disco, suelen estar comprendidos entre 4 y 64 sectores.

EJEMPLO 10.9

Si se tienen 512 B/sector y 4 sectores/clúster, una unidad de asignación se compone de 2 KB, con lo que toda la información transferida debe ser troceada en bloques de 2 KB. Si tenemos un fichero de 270 Bytes, se desperdiciarán 2 · 1024 − 270 = 1778 Bytes.

Según se observa en la Figura 10.9, los sectores de las pistas más exteriores son de mayor longitud que las interiores; ahora bien, el número de bits grabados en cada sector es siempre el mismo, con lo que la densidad de grabación (bits grabados por pulgada, bpi) será menor en las pistas exteriores que en las interiores. Esto es evidente si se tiene en cuenta que la velocidad de transferencia de información hacia, o desde, la superficie del disco es constante, con lo que, como el tiempo en recorrer un sector interior es igual al de uno exterior, en ambos casos se grabará la misma cantidad de información.

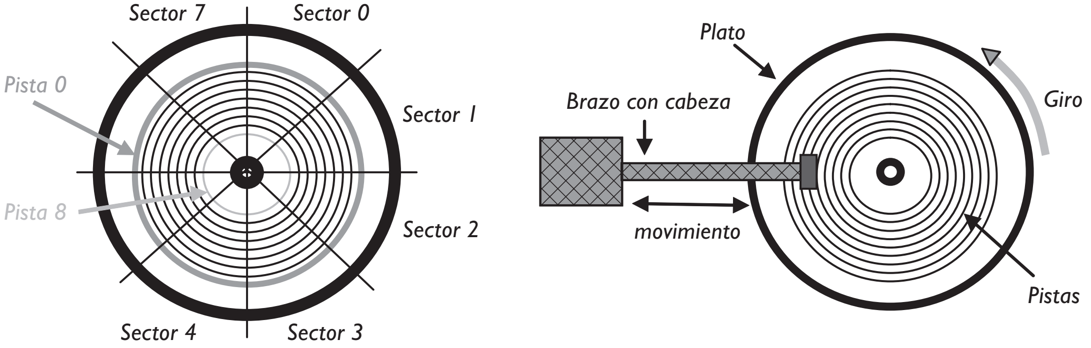


Figura 10.9. *(a) Distribución de sectores y pistas en la superficie de un disco; (b) El brazo se desplaza radialmente en busca de la pista a la que ha de acceder.*

La lectura y escritura en la superficie del disco, como se vio en la Sección 10.3.1, se hace mediante una cabeza o cápsula. La cabeza, en las unidades de cabeza móvil, está insertada en un extremo de un brazo mecánico móvil, que se desplaza hacia el centro o hacia la parte externa del disco bajo el control de los circuitos electrónicos del periférico (Figura 10.9b). El direccionamiento para leer o grabar un sector del disco se efectúa dando al periférico: el número de unidad, el número de superficie, el número de pista, y el número de sector. El brazo sitúa rápidamente la cápsula encima de la pista correspondiente y espera a que el sector en cuestión se posicione (como consecuencia del giro del plato) bajo la cápsula.

En el acceso (lectura o escritura) de un bloque de información de capacidad *C*, por ejemplo, hay que considerar tres operaciones, cada una consumiendo su tiempo correspondiente:

1. La cabeza debe posicionarse encima de la pista donde se encuentra el sector inicial del bloque a transferir. La duración de esta operación se denomina **tiempo de búsqueda (T_b)**, y puede suponerse que es lineal. Depende del tiempo T_0, que tarda en reaccionar el motor de desplazamiento una vez recibida la orden, del tiempo medio que tarda la cabeza en atravesar una pista, t_p, y del número de pistas, n_p, a recorrer hasta posicionarse en la buscada. Se verifica, por tanto:

   [10.5]
   $$T_b = T_0 + n_p \cdot t_p$$

2. La cabeza espera encima de la pista hasta que, como consecuencia del giro del plato, el sector a acceder se posicione debajo de ella. El tiempo que dura esta operación se denomina **tiempo de espera** (o **latencia rotacional**), T_e, y depende de la velocidad de rotación del disco, y de la posición donde se encuentre el sector a acceder. Puede ser que en el momento que llega la cabeza a la pista, el inicio del sector se encuentre justo debajo de aquélla, con lo que el tiempo de espera sería cero; pero también puede ser que acabe de sobrepasarla con lo cual el tiempo sería el que dure una rotación completa; es decir, $1/\omega_r$ siendo ω_r la velocidad de rotación del disco en revoluciones/segundo. Teniendo en cuenta todas las situaciones posibles, por termino medio el tiempo de espera será el que se tarda en realizar media rotación; es decir:

   [10.6]
   $$T_e = \frac{1}{2 \cdot \omega_r} \textit{ segundos}$$

3. A continuación hay que considerar y estimar el **tiempo de lectura** o **tiempo de escritura** de los *C* bytes que constituyen el bloque. Los circuitos electrónicos y buses deben ser tales que sean capaces de leer al ritmo con que se desplaza el plato bajo la cabeza lectora/grabadora; con lo que, si una pista tiene una capacidad de C_p bytes, la velocidad de lectura será:

   [10.7]
   $$v_t = C_p \cdot \omega_r \textit{ Bytes/segundo}$$

   Y el tiempo total de lectura de los *C* bytes será:

   [10.8]
   $$T_t = \frac{C}{v_t} = \frac{C}{C_p \cdot \omega_r} \textit{ segundos}$$

   Entonces, el tiempo conjunto (total de acceso y lectura/escritura) es:

   [10.9]
   $$T_c = T_b + T_e + T_t$$

Se denomina **tiempo de acceso**, T_a, al tiempo que tarda la unidad en posicionarse al inicio del sector al que se quiere acceder. Del razonamiento efectuado anteriormente se deduce que será la suma del tiempo de búsqueda y del tiempo de espera; es decir:

[10.10]
$$T_a = T_b + T_e$$

Podemos concluir que para leer un determinado bloque de información consecutiva hay que sumar el tiempo de acceso al inicio del bloque y el tiempo de transferencia. De lo dicho anteriormente se deduce una cuestión práctica de una gran relevancia: cuanto mayor es el bloque a leer menor será el tiempo efectivo total de lectura o escritura. En otras palabras, se tarda menos en leer un bloque de *C* bytes si se puede realizar con un solo acceso en vez de hacerlo con varios accesos.

En la mayor parte de los tipos de discos duros las cabezas de lectura/grabación nunca llegan a tocar la superficie del disco, retrayéndose el brazo cuando el disco no está en régimen de funcionamiento. Cuando el disco adquiere su estado estacionario de rotación se está en condiciones de leer/escribir, saliendo el brazo de su posición de reposo y posicionándose la cápsula encima de la superficie del disco, pero *flotando* sobre ella, sin llegar a tocarla. Esto ocurre a causa de que la gran velocidad de rotación genera turbulencias en la capa de aire próxima a la superficie, actuando como una almohadilla para el brazo haciéndole *volar* o *flotar*. La distancia entre la cabeza y la superficie es del orden de micras o decenas de micras. En el caso de que, por accidente, la cabeza toque la superficie (*aterrice*) se deteriora, e incluso puede que la cápsula se averíe. En los disquetes, por el contrario, durante las operaciones de lectura y escritura la cabeza entra en contacto físico con la superficie del plato.

Los platos de discos suelen tener una o varias referencias físicas (orificios o muescas) para poder identificar los sectores y pistas del plato. Esto se denomina **sectorización hardware o física.** En los disquetes sólo existe un orificio de alineamiento y referencia. Este orificio, cuando el disco gira, es detectado por un conjunto fotodiodo/fototransistor, utilizándose como punto de referencia para el acceso a las distintas pistas y sectores. Las unidades de discos duros suelen tener unas muescas que identifican los límites de cada sector y el primer sector de la pista.

Antes de utilizar un disco es necesario efectuar sobre él unas grabaciones para **dar formato** al disco. Al *formatear* un disco se definen por software las pistas, y sectores (pueden no coincidir con los sectores hardware); además se inicializa un **directorio** para información sobre el contenido del disco (el directorio es un *índice* del contenido del disco) (Sección 13.8). Cada sector tiene grabado su dirección, los datos del usuario (si los hay), e información de control que identifica los límites del sector, e incluye redundancias para detectar posibles errores en la grabación de los datos. En la Figura 10.10 puede verse un ejemplo de la información contenida en un sector formateado. El formateo efectúa una **sectorización software o lógica** que detecta, y elimina para ulteriores grabaciones, las zonas del disco deterioradas. También inicializa los sectores con la información antes indicada (graba dirección y controles, e inicializa el directorio).

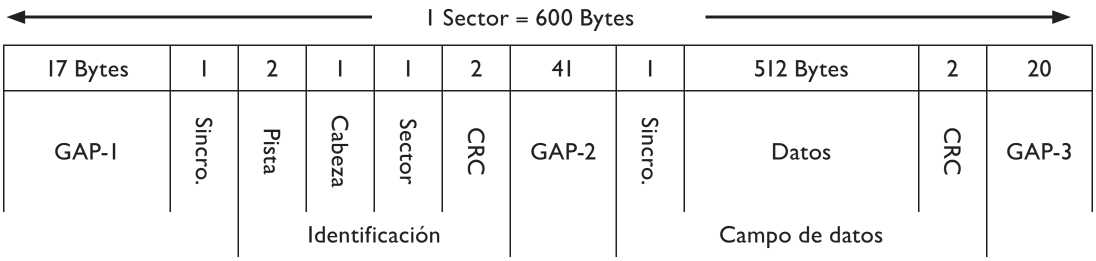


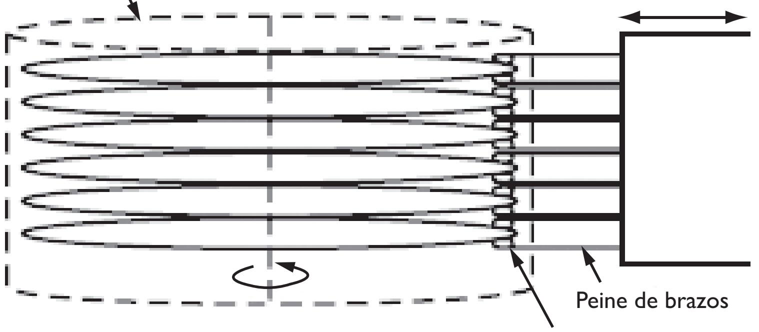
Figura 10.10. *Ejemplo de información contenida en un sector de un disco (GAP, es una zona de separación, sin grabar; y CRC son bits de un código detector de errores).*

Debido a la alta velocidad de giro del disco, con frecuencia los circuitos de control no pueden procesar consecutivamente la información de dos sectores adyacentes, por lo que, para reducir el tiempo de espera, los sectores se suelen direccionar (es decir, grabar) alternativamente (*interleaving*, véase Figura 10.25a). Por ejemplo en la figura, si se acaba de leer el sector 0, mientras el periférico procesa esta lectura la cabeza lectora habrá pasado a la parte central del sector 4, pudiéndose recuperar rápidamente la información del sector 1, después de la del 0 (la información se graba y recupera según las direcciones consecutivas indicadas en la figura).

Se pueden considerar cinco tipos de unidades de discos magnéticos:

- **Discos de cabezas fijas**. Tienen una cabeza individual de lectura/escritura por cada pista, consiguiéndose con ello tiempos de accesos relativamente bajos (del orden de milisegundos), ya que estos vienen fijados únicamente por la velocidad de giro del disco. Existían unidades con un único plato o con varios platos. Por ejemplo la unidad Data General 6006 disponía de 8 superficies, cada una de ellas con 128 pistas, cada pista contiene 8 sectores de 256 palabras (16 bits) cada uno. El tiempo medio de acceso a un sector era de 9,5 ms, y el disco giraba a 3.540 rpm. La velocidad de transferencia con el computador era aproximadamente de 500 KB/s.

a)



b)

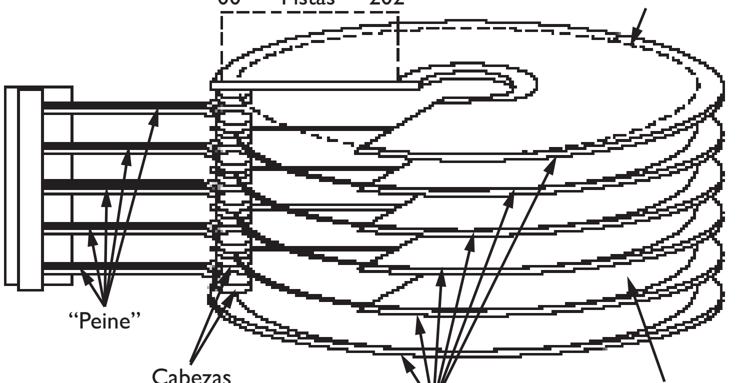


Figura 10.11. *(a) y (b) Esquemas simplificados de paquetes de discos.*

- **Paquetes de discos**. Son unidades compuestas por varios platos que giran solidariamente alrededor de un eje común (Figura 10.11). Las cabezas de lectura/escritura son móviles, existiendo una por superficie, y se desplazan simultáneamente a gran velocidad radialmente buscando la pista en que se encuentra el sector que deben leer o escribir. Cada cabeza lee/graba en el sector correspondiente a su superficie. Cada grupo de pistas de igual radio se denomina **cilindro**, existiendo tantos cilindros en el dispositivo como pistas en una superficie. Con frecuencia las superficies externas no se utilizan para grabar; así, una unidad con 6 platos puede utilizar sólo 10 superficies, y una de 11 platos, 20 superficies. Usualmente las superficies externas, como en el caso de la Figura 8.27b, no se utilizan para grabar; así una unidad con 6 platos puede utilizar sólo 10 superficies, y una de 11 platos, 20 superficies. En ocasiones una de las superficies contiene únicamente información para centrar las cabezas sobre las pistas. Esta superficie se suele denominar "superficie del servo", y el acceso y centrado a un cilindro se efectúa con un proceso de realimentación: se lee en la superficie del servo y se comprueba si la cabeza está o no centrada, corrigiéndose, en su caso, la posición. Los paquetes de discos no se utilizan en la actualidad, pero han servido de base para el desarrollo de la tecnología Winchester, que es la utilizada en los discos duros modernos.
- **Discos cartucho**. Son unidades con un plato y dos superficies de grabación, encerradas dentro de una carcasa, con una abertura lateral por donde se introducen las cabezas.
- **Disquetes**. Los disquetes son pequeños discos cuyos platos son flexibles, ya que están constituidos por un material plástico, Mylar, recubierto de óxido férrico. La velocidad de rotación de funcionamiento suele ser de 300 a 600 rpm, y son intercambiables. Los primeros disquetes eran de 8" y 5¼" y posteriormente tuvieron gran popularidad los de 3½". Tuvieron una gran relevancia debido a que eran un soporte intercambiable, de pequeño tamaño y con una capacidad típica de 1,44 MB. Hoy día han sido sustituidos por las memorias flash USB, mucho más capaces, rápidas y de menor tamaño (Sección 10.3.5).

Tabla 10.4. *Características de unidades de disco de tipo Winchester.*

Característica	Valores
Densidades de grabación	Radial: 800 a 2.000 pistas/cm Transversal: 50.000 a 100.000 bits/cm
Nº de cilindros	300 a 3.000
Nº de cabezas	2 a 256
Sectores/pista	8 a 64
Bytes/sector	512 Bytes
Tamaño plato	14'', 5¼'', 3½'', 2½'', 1,8'' y 1,3''
Tiempo de búsqueda	3 a 16 ms
Velocidad de rotación	3.600, 5.400, 7.200, 10.800 rpm
Velocidad de transferencia	3 a 320 MB/s
Método de grabación	RLL
Interfaces	ST506, ESDI, SCSI-2, SCSI-3, IDE y EIDE

- **Discos Winchester** (IBM, 1973) son paquetes de dos o más platos en los que, con objeto de reducir los efectos de la suciedad ambiental, están herméticamente cerrados y son fijos (no intercambiables). Las cabezas van más próximas a la superficie que las de las unidades descritas en los párrafos anteriores, lográndose grandes densidades de grabación. Las unidades de **discos duros** de hoy día son de tecnología Winchester, y pueden tener de 2 a 20 platos, girando a velocidades usualmente entre 3600 y 10.800 rpm, conteniendo de 500 a más de 100.000 pistas por superficie, 32 a 800 sectores/pista, y con dimensiones de los platos de 1,3 a 14”. En la Tabla 10.4 se muestra un resumen de las características de los discos duros. En la Figura 10.12 se muestra una foto de una unidad IBM Ultrastar 36ZX que contiene 10 platos que giran a 10.800 rpm, y es capaz de almacenar hasta 36 GB.



Figura 10.12. *Vista superior de la unidad de disco IBM Ultrastar 36ZX (© IBM Corporation).*

La consecución de las prestaciones actuales de los discos[1] (capacidades del orden de decenas de GB y tiempos de espera del orden de milisegundos) ha sido posible gracias a la evolución tecnológica que ha permitido abordar el perfeccionamiento de los discos en los aspectos que a continuación se indican.

[1] Los PC iniciales, del año 1981, tenían discos duros de 10 MB de capacidad, con un costo de 100 € por MB, en el año 2006 los había de 1 TB, con un costo menor de 0,0007 € por MB, lo que supone aproximadamente una mejora en capacidad del 10.000.000%, y una reducción del 99,9993% en el precio, en 25 años.

Velocidad de giro

Según se indicó anteriormente, expresión [10.6], el tiempo de espera, T_e, es inversamente proporcional a la velocidad de giro o rotación, ω_r; es decir, la unidad de disco será más rápida cuanto mayor sea la velocidad de giro. La velocidad tradicional de los discos duros es de 3.600 rpm; sin embargo, se han logrado obtener velocidades de giro mayores, como es el caso de la unidad de IBM citada anteriormente que alcanza 10.800 rpm, consiguiéndose así un tiempo medio de espera de 2,8 ms.

Por otra parte la corriente inducida en la cabeza lectora de una unidad de disco depende de la *variación* del flujo magnético que la atraviesa, con lo que si el disco se mueve más rápidamente las *variaciones* del flujo son mayores, lo que implica más sensibilidad en la detección de la información grabada en la superficie del disco. Similarmente la grabación de los campos magnéticos se puede realizar con corrientes menores.

La velocidad alta de giro presenta dos problemas: mayor probabilidad de avería en las superficies, ya que si las cabezas aterrizan sobre ellas el deterioro es casi seguro, y un aumento considerable de la temperatura. El aumento de la temperatura obliga a tener que realizar periódicamente, en intervalos de 2 a 10 minutos, un ajuste o recalibración térmica (**TCAL**, *Thermal recalibration*) de los elementos mecánicos. La TCAL es efectuada por el controlador de disco, e invierte en ella del orden de medio segundo, durante el cual no es posible leer o escribir en el disco.

Plato

Tradicionalmente el plato (sustrato de las superficies magnetizables) es de aluminio con un diámetro de 3½", o 2½" en los portátiles. Diversas empresas, como Areal y Maxtor, han desarrollado platos de cristal. La principal ventaja de este material sobre el aluminio es que es más ligero y rígido. Estas propiedades, junto a lo liso de sus superficies, lo hacen más seguro frente a posibles impactos con las cabezas (muy adecuado, por tanto, para equipos portátiles), y también su ligereza hace que el motor de giro consuma menos energía que para mover platos más pesados (de aluminio). El diámetro de las unidades comercializadas con plato de cristal es de 2½".

Grabación de las pistas

Hay tres formas de grabación de discos:

- **CAV: velocidad angular constante**. Es la forma tradicional de grabación, que se realiza a una velocidad de escritura y lectura de las pistas (en bits/segundo) constante, de forma que, como la velocidad de rotación también es constante, las pistas más internas se graban con una densidad mayor que las externas. De esta forma el número de bits grabados en cada sector es siempre el mismo, con lo que la densidad de grabación (bits/pulgada o bpi) es menor en las pistas exteriores.
- **ZCAV** (o **MZR**) **velocidad angular constante por zonas** o **ZBR** (*Zoned-bit recording*). En los discos más avanzados, para conseguir mayor capacidad, la velocidad de grabación de las pistas exteriores se hace mayor, de forma que en las pistas más largas la capacidad de datos resulta mayor (se graban más sectores). De esta forma se consigue una densidad de grabación uniforme y que en las pistas externas haya más sectores que en las internas. Esta forma de grabación (que únicamente complica la electrónica del controlador, ya que los elementos electromecánicos no varían) tiene dos consecuencias prácticas relevantes: (a) se consigue una mayor capacidad de almacenamiento, y (b) como la velocidad de giro sigue manteniéndose constante, a los datos de las pistas más externas se accede 2 ó 3 veces más rápidamente que a los de las pistas internas. Con el **sistema ZCAV** las pistas contiguas se agrupan por zonas (8 a 30 zonas/disco). En los accesos a las zonas

más exteriores se incrementa la velocidad angular, con lo que el número de sectores por pista se incrementa de las pistas interiores hacia las exteriores. Se obtiene un rendimiento óptimo del disco ubicando en los cilindros más externos la información que más se intercambia con el computador (los archivos de intercambio de la memoria virtual, véase Sección 9.4.5, los ficheros temporales de respaldo de seguridad, etc.).

- **CLV: velocidad lineal constante:** las pistas más externas contienen más información que las internas, rotando el disco más rápidamente en ellas. Este sistema es el que se utiliza en los discos ópticos (Sección 10.3.4).

Controladores

El controlador del disco suele contener un microprocesador específico que admite órdenes para controlar tareas tales como: *arranque, leer, escribir,* y *dar formato*, monitoriza el movimiento del peine, detecta y corrige errores, y convierte los bytes (información que llega en paralelo) en patrones de grabación (RLL, serie). Cada vez se utilizan controladores de disco más inteligentes (Figura 10.13). Si se desea una gran compatibilidad, y es para utilizar en equipos pequeños (PC y estaciones de trabajo), se utilizan preferentemente unidades IDE (también denominadas **ATA**, *AT Attachement*). Formalmente una unidad **IDE** es aquella que integra en su interior el controlador, de forma que puede conectarse directamente al bus del sistema (ISA, EISA o bus local, por ejemplo). Para equipos mayores, se utilizan controladores para bus SCSI (Capítulo 12). Esta interfaz es más flexible y de mejores prestaciones que la IDE. Existen circuitos integrados para adaptar un bus SCSI a un bus de 16 bits ISA (por ejemplo, el circuito Adaptec 1542), o a un bus de 32 bits EISA (por ejemplo, el circuito Adaptec 1742). En resumen, existen diversas técnicas para interconectar una unidad de disco a un computador, tal y como se indica en la Figura 10.13. En el Capítulo 12 se describirán brevemente las interfases y buses citados en la figura.

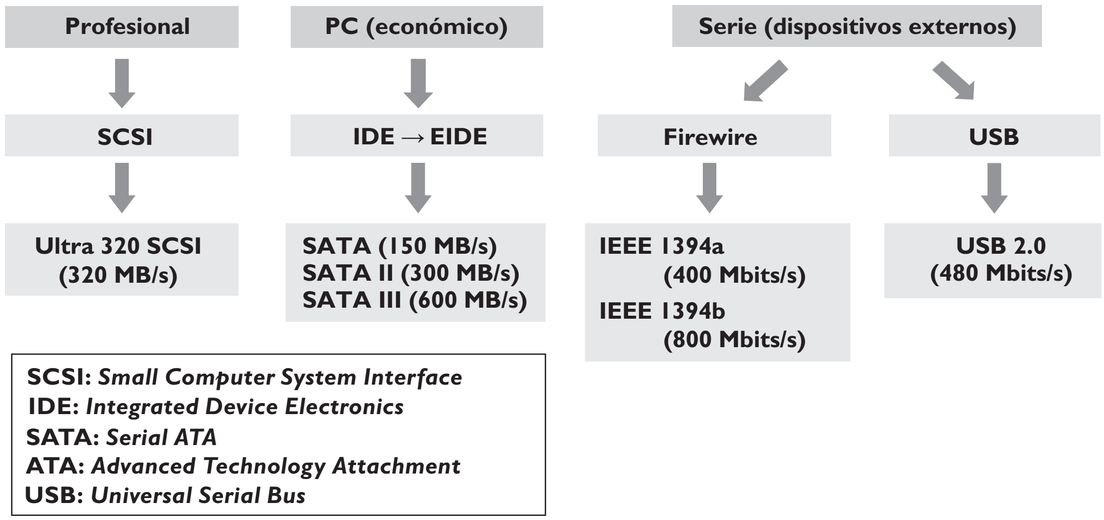


Figura 10.13. *Interfases o buses usualmente utilizadas con discos magnéticos.*

Buffer

Las unidades de disco, como la mayoría de periféricos, disponen de una memoria intermedia (*buffer*) en el controlador correspondiente, a través de la cual se realizan las transferencias

con el procesador o la memoria principal. La velocidad media de transferencia interna de datos (ancho de banda interno) viene dada por el número de bytes transferidos por segundo desde el disco al *buffer*. El *buffer* permite adaptar los diferentes anchos de banda entre el bus interno y el bus externo. Por ejemplo, una unidad de disco puede tener un ancho de banda interno de 95 MB/s y estar conectada a un bus o interfaz SATA/150 con ancho de banda de 150 MB/s.

Caché

Las unidades de disco modernas disponen de la posibilidad de efectuar la escritura en una memoria caché en el controlador de discos. Esta facultad hace que la velocidad efectiva del periférico mejore considerablemente. El tamaño de la memoria caché puede llegar a decenas de MBytes en los discos de mayor capacidad; de esta forma en una caché de 8 MBytes se pueden almacenar hasta 16.384 sectores. El controlador, simultáneamente a ir recibiendo información a través del bus de E/S, va grabando la información de la caché en las superficies del disco. Así, además de obtener mayor rendimiento del procesador, la velocidad efectiva de escritura del disco es mayor ya que la información recibida en sucesivas operaciones se podrá grabar en sectores contiguos sin necesidad de consumir los tiempos de latencia inherentes a cada operación individual de lectura/escritura. También la caché permite aprovechar el principio de localidad espacial efectuando una **lectura anticipada de datos** con la que en la caché no sólo se almacena el sector solicitado, sino además todos los que le siguen (o una pista o cilindro completo, por ejemplo). Cuando se trata de leer un bloque de datos, primero se consulta si están en la caché y si así es se captan de ella, sólo en caso contrario se realiza la lectura de las superficies del disco.

EJEMPLO 10.10

Como ejemplo de unidad de disco duro (paquete Winchester) para un servidor de archivos se puede citar el Seagate Barracuda 7.200,8, que tiene las siguientes características:

- *Capacidad: 372 GB.*
- *Tiempo medio de búsqueda: 8 ms.*
- *Latencia rotacional media: 4,16 ms.*
- *Velocidad de giro: 7.200 rpm.*
- *Velocidad máxima de transferencia externa: 150 MB/s.*
- *Velocidad de transferencia interna: 95 MB/s.*
- *Bytes/sector: 512.*
- *Número de sectores: 781.250.000.*
- *Caché: 8 ó 16 MB.*

10.3.2.2 Unidades RAID

Una unidad ***RAID*** (*Redundant Array of Independent Disks*) o ***agrupación redundante de discos independientes***[2], es un conjunto de discos que funcionan en paralelo y que son considerados por el sistema operativo como una única unidad. El objetivo de este tipo de unidades es doble:

- aumentar la velocidad, y
- mejorar la seguridad y fiabilidad de los datos almacenados.

[2] RAID inicialmente eran las siglas de *Redundant Array of Inexpensive Disks* o agrupación redundante de discos de bajo coste.

La idea básica consiste en almacenar los datos en varios discos que funcionan en paralelo (Figura 10.14). El sistema operativo considera divididos los datos del archivo a almacenar (o leer) en *tiras* consecutivas, cada una de las cuales corresponde a un número determinado de sectores. Supongamos que cada tira es de 2KB, y corresponde a 4 sectores; y que la unidad RAID se compone de 5 unidades de discos (D0 a D4). Entonces, si el archivo a almacenar es de 30 KB, se tienen en total 15 tiras (T0, T1, ..., T14). El sistema operativo da la orden a los controladores de almacenar la tira T0 en el disco D0, la T1 en el D1, la T2 en el D2, la T3 en el D3, la T4 en el D0, ..., y la T15 en el D3. Obsérvese que el disco D4 no se utiliza para almacenar los datos originales; en efecto, en el disco D4 se almacenan bits de comprobación. Concretamente se crea una tira de paridad, TP(0-3) en la que cada uno de sus bits corresponde al bit de paridad de los cuatro bits del mismo orden correspondientes a T0, T1, T2 y T3. La segunda tira, TP(4-7) almacenada en el disco D4 corresponderá a los bits de paridad de las tiras T4, T5, T6 y T7; y así sucesivamente.

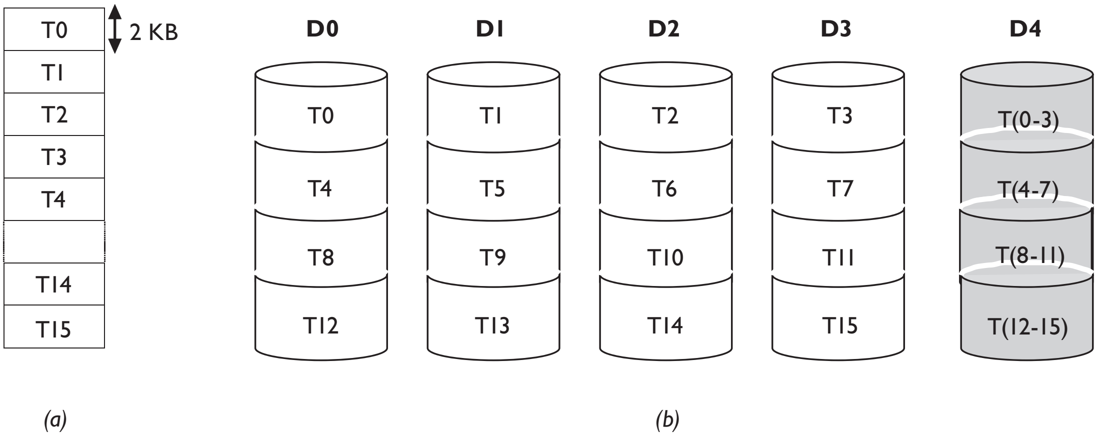


Figura 10.14. *Ejemplo de unidad RAID de nivel 4.*

Con este ingenioso esquema la velocidad se aumenta ya que, al funcionar en paralelo la velocidad de transmisión efectiva aumenta notablemente; así, si la velocidad de transmisión de un disco individual es de 40 MB/s, la memoria puede enviar datos al conjunto RAID a 160 MB/s. La fiabilidad del sistema aumenta considerablemente, ya que aparte de las redundancias de comprobación que se incluyen dentro de cada sector, existe un disco con redundancias adicionales. Si se avería una unidad cualquiera, la D2, por ejemplo, el sistema operativo, a partir del resto de los discos (incluyendo el D4), recalcula la información perdida en D2 y la almacena en el disco redundante (D4) (véase Ejercicio 8.25). Una vez hecha esta operación, el disco D4 pasa a sustituir al D2, accediéndose a la información de los discos en el siguiente orden D0, D1, D4 y D3; de forma que el sistema puede seguir funcionando normalmente. En el momento de tener reparado y listo de nuevo el disco D2, el sistema operativo, vuelca toda la información de D4 en D2, y recalcula los bits de paridad para almacenarlos en D4; para prevenir futuras averías.

Existen diversas alternativas para organizar las unidades RAID, que se suelen conocer como niveles. La alternativa anteriormente explicada se conoce como RAID nivel 4. Aquí únicamente hemos pretendido que el lector comprenda el concepto básico sobre estas unidades, y si esta más interesado en este tema puede encontrar más información en las referencias [Sta00, Tan99].

10.3.3 Cintas magnéticas

Las cintas magnéticas se basan en los mismos principios de lectura/grabación de las cintas de los magnetófonos y casetes convencionales. El soporte de grabación consiste en un plástico (poliéster) muy flexible recubierto de un óxido magnetizable (óxido de hierro, óxido de cromo, etc.) de aproximadamente 100 μm de espesor. La cinta se encuentra enrollada, y la lectura y grabación se efectúa haciéndola pasar por una estación de lectura/escritura al transferirla de un carrete en un eje de giro a otro carrete en otro eje. En las primeras décadas del uso de computadores las **cintas magnéticas de tipo carrete** se utilizaban como único sistema de memoria masiva, en la actualidad han sido sustituidas por cintas de tipo casete o cartucho. Las cintas magnéticas constituyen un soporte de información barato y de gran capacidad, pero son muy lentas (acceso secuencial). Hasta que se popularizaron y normalizaron los disquetes, la cinta magnética era el medio idóneo para intercambiar programas y datos entre servicios de informática; en la actualidad estos cometidos son realizados a través de Internet o por medio de discos ópticos (CD o DVD). Esto fue así porque los métodos, formatos y códigos de grabación de cintas se normalizaron rápidamente. En la actualidad, la principal misión de las cintas magnéticas es obtener **copias de seguridad** de la información contenida en discos completos, o almacenar información obsoleta (ficheros *históricos*). Existen gran cantidad de tipos y tecnologías de unidades de cintas magnéticas; los más importantes se enumeran en la Figura 10.15.

- *Cintas clásicas o de carrete* (cintas ½" de ancho). Grabación lineal paralela de 7 ó 9 pistas.
- *Cartuchos con cintas* de ¼", ½" u 8 mm.
- *Cartuchos compactos* (cartuchos pequeños):
  - QUIC (***Q**uarter **I**nch **C**artridge*).
  - DAT (***D**igital **A**udio **T**apes*).
  - EXABYTE.
  - SAIT (***S**uper **A**dvanced **I**ntelligent **T**ape*).
  - DLT/SDLT (***D**igital **L**inear **T**ape/Super DLT*).
  - LTO (***L**inear **T**ape **O**pen*).

Figura 10.15. *Principales tipos y tecnologías de unidades de cintas magnéticas.*

La grabación de una cinta se hace en unidades de información denominadas **bloques físicos** o **particiones** que contienen un conjunto de bytes de una longitud preestablecida, de forma similar a los sectores de los discos magnéticos. En el caso de las cintas clásicas de carrete la longitud del bloque podía ser seleccionada arbitrariamente por el usuario dentro de unos límites (usualmente entre 200 y 1.024 Bytes). La cinta iba leyendo o escribiendo bloque a bloque, que cargaba en su memoria intermedia. Cuando acababa de escribir en la cinta el bloque contenido en la memoria intermedia, la cinta se detenía, esperando a que el computador enviase el siguiente bloque (un proceso similar tiene lugar durante la lectura), para volver a repetir el proceso de escritura bloque a bloque. Debido a que la cinta no puede detenerse instantáneamente, entre cada dos bloques consecutivos se desperdicia (no se graba) un determinado espacio (de ½ a ¾ de pulgada en las cintas de carrete) que se denomina **interbloque** o **IRG** (*inter-record-gap*). Cada bloque contiene, además de los datos del usuario, secuencias preestablecidas de caracteres y espacios identificadores de los límites del bloque, e información adicional redundante para poder detectar automáticamente posibles errores de grabación.

En las cintas actuales es habitual que se disponga de dos modalidades de funcionamiento: de arranque/parada (o *start/stop*) y de bobinado continuo (*streaming*). Las primeras intercambian bloques de información de longitud similar a las cintas tradicionales, y las segundas intercambian bloques de gran longitud consiguiéndose aprovechar la cinta en aproximadamente un 95% o más (apenas hay IRG).

La capacidad de una cinta, *C*, depende fundamentalmente de su longitud, *L*, densidad de grabación, *d*, longitud de bloque y formato de grabación. Conforme han ido pasando los años y mejorando la tecnología las densidades de grabación se han incrementado desde unos 200 bpi a 160.000 bpi. Un cálculo aproximado, por exceso, de la capacidad puede efectuarse multiplicando longitud por densidad:

[10.11]

$$C \leq L \cdot d$$

Cuanto menores sean los bloques menos datos del usuario podrán almacenarse en la cinta, ya que por bloque se pierden los caracteres de identificación del bloque, los caracteres redundantes de detección de errores y el espacio IRG. No obstante, aplicando la fórmula anterior se puede obtener una idea de la capacidad máxima de una cinta. También conociendo la velocidad lineal de lectura/escritura, que se suele dar en pulgadas/segundo (ips o i/s) se puede determinar fácilmente la velocidad máxima de transferencia de datos entre la cabeza y la memoria intermedia (o computador).

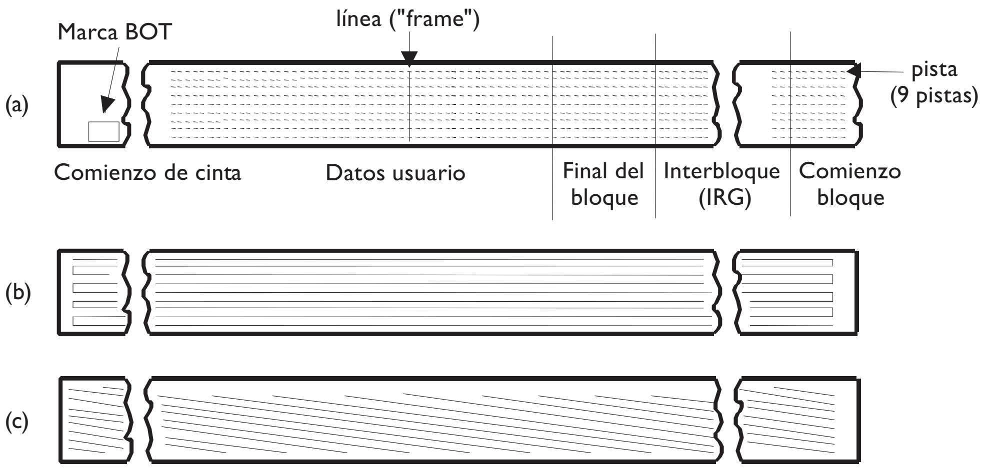


Figura 10.16. *Técnicas de grabación de una cinta: (a) todas las pistas en paralelo; (b) en serpentina, una única cabeza graba todas las pistas; y (c) en espiga, con cabeza móvil de barrido helicoidal.*

Existen tres técnicas básicas de grabar una cinta magnética (Figura 10.16), y cada tipo de unidad de cinta utiliza uno de ellos:

- **Lineal paralelo**. Se graba la información simultáneamente en varias pistas (Figura 10.16a). El conjunto de bits que se leen simultáneamente se denomina línea de grabación (*frame*), y en las cintas tradicionales corresponde a un carácter (de 6 u 8 bits) con un bit adicional de paridad, siendo, por tanto, las cabezas de 7 ó 9 pistas (es decir, la cabeza contiene 7 ó 9 bobinas de lectura/escritura, respectivamente). Actualmente los cartuchos que utilizan esta técnica de grabación suelen tener de 2 a 8 pistas.

- **Lineal serpentina**. Las unidades de cartuchos más utilizadas son de 24 pistas que se graban en forma de *serpentina* y en un instante dado sólo se graba una pista (Figura 10.16b). La cabeza lectora/grabadora puede desplazarse verticalmente en 24 posiciones preestablecidas (en la Figura 10.16b sólo se explicitan 9). La cinta se desplaza horizontalmente, y cuando comienza a grabarse, la cabeza se ubica en la posición superior, continuando en ella hasta que se llega al final de la cinta, en cuyo momento la cabeza desciende hasta la posición inmediatamente inferior. Cuando se llega al otro extremo de la cinta, vuelve a descender la cabeza, y así sucesivamente.
- **Helicoidal.** El tambor de lectura/grabación gira, de forma que hay dos movimientos superpuestos, el de la cinta que se desplaza de un eje de enrollamiento al otro, y el de la cabeza. De esta forma se obtienen velocidades relativas de desplazamiento entre tambor y cinta relativamente bajos, consiguiéndose mayor calidad de grabación y duración de la cinta. La posición del tambor y la disposición en él de las bobinas de lectura/escritura hacen que las pistas se graben en "forma de espiga", formando un ángulo de 6 grados con el eje longitudinal de la cinta (Figura 10.16c).

Las cintas magnéticas se pueden clasificar en los siguientes tipos (Figura 10.15):

- **Cintas clásicas o de carrete.** Son unidades para cintas de ½" de ancho, y grabación lineal paralela de 7 ó 9 pistas.
- **Cartuchos.** Existen diversos tipos de unidades de cartucho, como:
  - Con cintas de ¼" o de ½" de ancho. Ejemplo de ½", Magstar, con grabado lineal, 128 pistas o más, y capacidades de hasta 20 GB.
  - Con cintas de 8 mm. Ejemplo Magstar MP (*Magstar* ***M****ulti****P****urpose*), grabado lineal. Utiliza cartuchos tipo casete de 5GB, y estas unidades se pueden configurar en forma de conjuntos o bibliotecas donde las cintas son manipuladas por robots.
- **Cartuchos compactos** (cartuchos pequeños):
  - QUIC. El nombre de esta tecnología se debe al tamaño de sus primeras cintas. Básicamente hay dos tipos: SLR (de 5¼") y Travan (de 3½"). Las capacidades varían desde 40 MB a 25 GB.
  - DAT. Cintas de 4mm de grabación de barrido helicoidal, con velocidades relativas entre cinta y tambor de 0,32 i/s. Ejemplo DAT DDS-5 de 40 GB. Grabación digital.
  - EXABYTE. Cintas de 8mm de barrido helicoidal con velocidades relativas entre cinta y tambor 30 i/s. Grabación analógica. Capacidad 10 GB.
  - SAIT. Cintas de ½", grabación helicoidal, con capacidades de 500 GB (SAIT-1) y 1 TB (SAIT-2). Grabación helicoidal. En estas cintas la capa magnetizable se forma vaporizando metal con cobalto utilizando técnicas avanzadas, consiguiéndose así muy grandes densidades de grabación (155.000 BPI en SAIT 1), velocidad de transferencia 30 MB/s.
  - DLT/SDLT. Cintas de ½", 128 ó 208 pistas lineales con capacidades de 40 a 80 GB, y anchos de banda de 3 a 8 MB/s. Con las unidades SDLT se consiguen capacidades de 110 a 300 GB con velocidades de transferencia de 11 a 36 MB/s.
  - LTO. Estándar que define componentes tales como cabeza lectora/grabadora, trazado de pistas y tecnología de servomotores. Hay dos formatos (1998): Accelis (25 GB) y Ultrium (100 y 200 GB).

Los parámetros sobre capacidades y anchos de banda dados anteriormente corresponden a **datos nativos** (sin comprimir); con técnicas de compresión se consiguen capacidades y velocidades mayores.

EJEMPLO 10.11

Las especificaciones de la tecnología SAIT-1 son las siguientes:

- *Método de grabación: helicoidal.*
- *Densidad de grabación: 155.000 bpi.*
- *Longitud del boque: fijo o variable.*
- *Capacidad 500 GB (con compresión ALDC 2,6:1, 1,3 TB).*
- *Velocidad de transferencia sostenida: 30 MB/s.*
- *Velocidad de transferencia máxima en ráfagas (síncrona SCSI): 160 MB/s.*
- *Tiempo medio de carga: 23 s.*
- *Tiempo de acceso medio: 70 s.*
- *Velocidad máxima de búsqueda: 394 pulgadas/s.*
- *Velocidad máxima de rebobinado: 551 pulgadas/s.*
- *Velocidad rotacional: 5.000 rpm.*
- *Tamaño de memoria intermedia: 75 MB.*
- *Longitud de la cinta: 600 m de cinta de ½".*

10.3.4 Discos ópticos

Los discos ópticos son dispositivos para almacenamiento masivo de información, cuya lectura se efectúa por medios ópticos. Existen diversos tipos, siendo los más relevantes:

- **CD** (*Compact Disk*) o **disco compacto.**
- **DVD** (*Digital Versatil Disk*[3]) o **disco digital versátil**.

Las características comunes de estos sistemas son:

- alta capacidad de almacenamiento, típicamente entre 650 MB y 17 GB; es decir, en este último caso equivalente a unos 12.000 disquetes de 1,44 MB;
- el precio por bit es el más bajo de todos los dispositivos de memoria masiva;
- los soportes de grabación (los discos) son intercambiables (como los disquetes), y son, aproximadamente, unas 10 veces más lentos que los discos duros y el doble de rápidos que los disquetes; y
- la degradación o pérdida de información es prácticamente nula, ya que no se producen desgastes por lectura, y no necesitan altos requerimientos en la limpieza de sus superficies externas.

En la mayoría de los discos ópticos la información, a diferencia de las unidades de disco magnético, es grabada en espiral (y no en circunferencias concéntricas), y puede ser leída (dependiendo del tipo de unidad) a velocidad lineal constante (CLV), o a velocidad angular constante (CAV), como las unidades tradicionales de discos magnéticos (microsurcos). Tal como indicamos en la Sección 10.3.2.1, en las unidades CLV la velocidad de rotación depende de la posición radial de la cabeza (en este caso de 600 a 1.800 rpm), consiguiéndose directamente así que la densidad de grabación sea constante. A continuación se describe brevemente cada uno de los tipos de estos dispositivos.

[3] Inicialmente se denominaban *Digital Vídeo Disk,* Disco Digital para Vídeo; al extenderse su uso a otras aplicaciones se ha cambiado el significado de las siglas.

10.3.4.1 Discos compactos (CD)

Existen tres tipos de discos compactos: CD-ROM, CD-R y CD-RW, que a continuación se describen brevemente.

CD-ROM

Los **CD-ROM** (*Compact Disc, Read Only Memory*) son dispositivos de sólo lectura, ya que el proceso de grabación resulta muy complejo, siendo de interés comercial sólo cuando se producen tiradas grandes de discos con la misma información. Esto hace que sus aplicaciones principales sean para grabar, por ejemplo, enciclopedias accesibles con computador, grandes manuales de computador, distribución de sistemas operativos, etc.

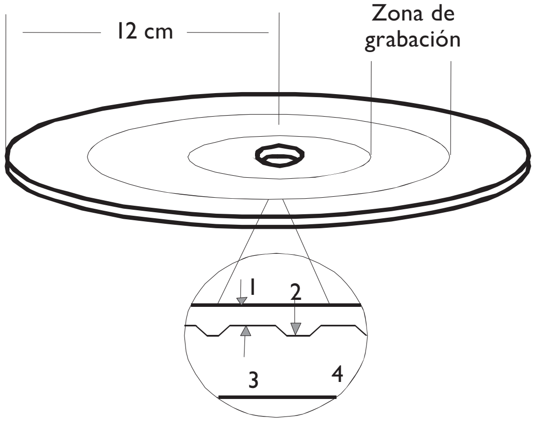


Figura 10.17. *Estructura básica de un disco óptico.*

La información es almacenada en forma de **hoyos** (*pits*) y **valles** (*lands*), grabados mecánicamente sobre un substrato de aluminio brillante, y es leída midiendo la luz de un haz láser reflejada sobre la superficie de hoyos y valles. En la Figura 10.17 puede verse un esquema simplificado de un CD-ROM. De la parte superior a la inferior se encuentra: la etiqueta, una capa protectora, la capa de aluminio brillante, en cuya superficie inferior se han grabado los hoyos y valles, y la base de plástico transparente (policarbonato). La lectura (según se muestra en la Figura 10.18a) se efectúa por abajo.

Para grabar un CD-ROM la información binaria de datos se transforma con un código especial denominado **código de canal**, con el que cada byte (carácter ASCII) queda representado por 14 bits. Esta información se graba en una cinta magnética, que es leída por un grabador de moldes (o *máster*). La información leída controla un modulador óptico de un láser de potencia. El haz láser es enfocado por medio de una lente sobre la superficie fotosensible que recubre el disco molde (que es de cristal). Mediante un proceso similar al seguido en una de las fases de construcción de circuitos integrados, se revela el disco, y las zonas de la superficie fotosensible que han sido expuestas a la luz se convierten en hoyos, resultando así un relieve exactamente igual al que tendrán los discos CD-ROM. Del molde original se obtienen varios discos denominadas matrices, que son copias en negativo de los discos definitivos. Con las matrices se obtiene la capa de aluminio de los CD-ROM, que posteriormente se aísla del exterior emparedándola entre la capa protectora superior y la base inferior de plástico.

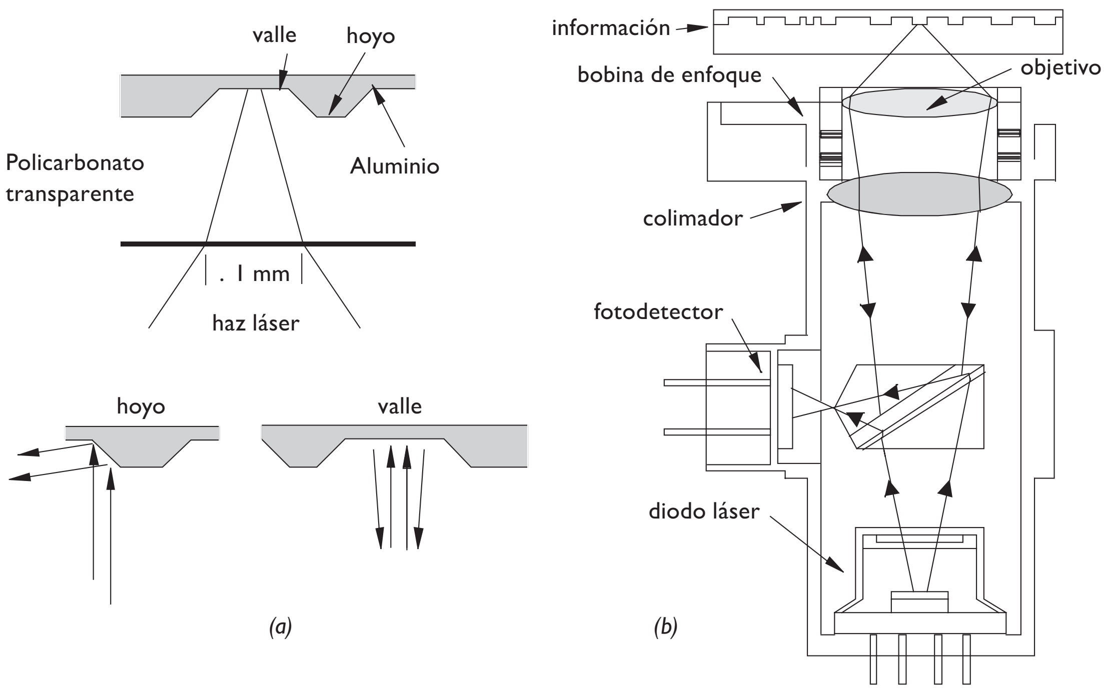


Figura 10.18. *(a) El haz es desviado al reflejarse en los bordes de los hoyos, y se refleja perpendicularmente en las superficies planas; (b) esquema de una cabeza lectora de una unidad CD-ROM.*

Los hoyos sobresalen de los valles 0,12 µm y tienen 0,6 µm de profundidad (espesor del Aluminio). La longitud a lo largo de pista de los hoyos y valles varia de 0,9 a 3,3 µm. La separación radial entre dos pistas consecutivas es de 1,6 µm, con lo que se obtiene una densidad de 16000 pistas/pulgada (t/i), muy superior a la de los discos magnéticos (los disquetes 96 t/i).

Para efectuar la lectura en un CD-ROM (al igual que en los otros tipos de videodiscos) se utiliza un haz de luz generado por un diodo láser de arseniuro de galio, que una lente enfoca sobre la superficie de la base de plástico, originando en ella un punto del orden de 1 mm de diámetro (Figura 10.18b), que a su vez es enfocado por la base de plástico (de 1,2 mm de espesor) en la superficie brillante de aluminio convirtiéndose en un punto de 1µm. Debido a las pequeñas dimensiones del haz de luz (del orden de la longitud de onda de la propia luz), la óptica de enfoque y de reflexión debe ser de una gran precisión. Ya que el enfoque del haz no se realiza sobre la superficie externa de la base de plástico, sino sobre el aluminio (Figura 10.18b) las posibles partículas de polvo o rayones en el exterior del disco apenas afectan a la lectura, como por el contrario ocurre en las unidades magnéticas.

La lectura se efectúa con un fotodetector que mide la cantidad de luz reflejada (genera una corriente proporcional a la intensidad de radiación incidente). Cuando la luz incide sobre los bordes de los hoyos se desvía (Figura 10.18a) de forma que apenas vuelve nada hacia el objetivo, obteniéndose así un "1". Por el contrario, cuando el haz incide sobre las superficies planas de hoyos y valles, se refleja casi sin desviación captándose en el detector prácticamente la totalidad de la intensidad del haz incidente, obteniéndose así un "0".

Codificación de la información en CD-ROM

Para analizar cómo se almacena la información en un CD-ROM hay que considerar dos aspectos: (1) el formato lógico y (2) el formato físico de grabación o **código de canal**. Los formatos lógicos más usados son el ISO 9660 (para computadores con MS-DOS) y el HFS (*Hierarchical File Systems*, Tabla 10.5) una variante del cual es usado en el Apple Macintosh). El formato físico o código de canal más usual es el EFM (modulación de ocho a catorce).

Desde el punto de vista del formato lógico la información se organiza en **bloques** (**sectores**). Cada sector contiene 2.352 Bytes (Figura 10.19):

- 12 bytes de sincronización:
  - 1 byte de ceros, 00_H
  - 10 bytes de unos, FF_H
  - 1 byte de ceros, 00_H
- 4 bytes de identificación (ID): minuto, segundo, bloque, modo.
- 2048 bytes de datos del usuario.
- 288 bytes de detección y corrección de errores:
  - 4 bytes de detección de errores (EDC)
  - 8 bytes todo ceros, 00_H
  - 276 bytes de corrección de errores (ECC).

1 Sector = 2.352 Bytes

1	10	1	1	1	1	1	2.048	288
00	FFFFFF	00	Minuto	Segundo	Sector	Modo	Datos	ECC
Sincronización			*Identificación*				*Datos*	*Detec. Error*

Figura 10.19. *Estructura lógica de un bloque o sector de un CD-ROM.*

Desde el punto de vista físico, según se ha indicado anteriormente, los "unos" se representan por los cambios de hoyo a valle o de valle a hoyo, los "ceros" se representan en las superficies de los valles y de los hoyos (cada 0,3 μm de longitud del hoyo o del plano, a lo largo de la pista, corresponde a un 0) (Figura 10.20). Como la longitud mínima físicamente obtenible para los hoyos y canales es de 0,6 μm, dos "unos" consecutivos deben estar separados al menos por dos "ceros". La serie máxima es de 11 ceros (33,3 μm/0,3 μm = 11). Como consecuencia de lo anterior cada byte (8 bits) es codificado con 14 bits de canal. La Tabla 10.5 muestra una parte del código EFM, que se utiliza con este objetivo.

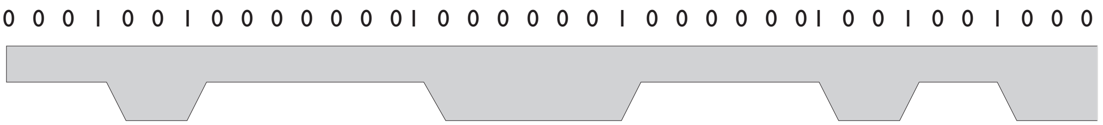


Figura 10.20. *Las superficies de zonas planas y hoyos corresponde a "ceros", y los bordes a "unos".*

A nivel físico los 2.352 bytes de cada sector se estructuran en tramas. Cada trama contiene 24 bytes de sector. Para conseguir una grabación adecuada, a continuación de cada

byte (14 bits de canal) se añaden 3 bits de separación o "refundición", de forma que para grabar 8 bits de datos de sector se utilizan 14 + 3 = 17 bits de canal o "símbolo CD-ROM". La trama finaliza con un patrón de sincronización física compuesto de 24 bits de canal y 3 de refundición, un símbolo de control y 8 símbolos de corrección de errores.

De acuerdo con lo anterior un sector quedará representado físicamente por 98 tramas.

Tabla 10.5. *Parte del código de modulación de ocho a catorce (EFM).*

Byte	Bits de canal
0000 0000	01 0010 0010 0000
0000 0001	10 0001 0000 0000
0000 0010	10 0100 0010 0000
0000 0011	10 0010 0010 0000
0000 0100	01 0001 0000 0000
0000 0101	00 0001 0001 0000
0000 0110	00 0100 0010 0000
0000 0111	00 1001 0000 0000
0000 1000	01 0010 0100 0000
0000 1001	10 0000 0100 0000
0000 1011	10 0100 0100 0000
etc.	etc.

EJEMPLO 10.12

Calcular el número de bits de canal que se leen por segundo y la velocidad de transferencia media de datos de usuario, en una unidad de CD-ROM estándar, sabiendo que la velocidad de lectura es de 75 sectores por segundo.¿Qué capacidad de datos cabría en un CD-ROM de 74 minutos?

Primero se puede calcular el número de bits de canal que contiene cada trama. Una trama está compuesta por:

(24 bytes/trama × 17 bits de canal/byte) + (27 bits patrón de sincronización) + (17 bits/símbolo × 1 símbolo de control) + (17 bits/símbolo × 8 símbolos de corrección de errores) = 588 bits/trama

Un sector contiene:

588 bits/trama × 98 tramas/sector = 57.624 bits de canal

Es decir, se leen:

57.625 × 75 = 4.321.800 bits/s

Como en cada sector hay 2KB datos de usuario; la velocidad de transferencia será:

2 KB/sect × 75 sect/seg = 150 KB/s

En 74 minutos se pueden leer:

74 × 60 × 75 = 333.000 sectores

es decir,

333.000 × 2KB = 666.000 KBytes

con lo que la capacidad máxima de la unidad es de 650 MBytes.

Otros CD

Según se indicó anteriormente hay otros dos tipos de CD:

- **Discos compactos grabables**, CD-R (*CD Recordable*). Son unidades de discos ópticos similares a los lectores de CD-ROM, pero que contienen un láser de mayor potencia de forma que en la propia unidad se puede efectuar la grabación del disco. Por tanto, con una de estas unidades el propio usuario puede grabar (una sola vez) el disco. La capa reflectante es de oro en lugar de aluminio, y está recubierta de una capa de tinte (similar a las utilizadas en fotografía). Inicialmente, la capa de tinte es transparente (**fase cristalina**), pasando a través de ella el láser, y pudiéndose reflejar sin problema en la capa de oro. Durante la grabación se hace incidir el haz láser con una mayor potencia (8 a 16 mW) que el de lectura (0,5 mW) en determinados puntos, quemándose el tinte y eliminándose su transparencia (pasa de fase cristalina a **fase amorfa**). Los puntos negros son interpretados como transiciones pozo-valle, o viceversa. Este método se denomina **sistema de grabación por cambio de fase**.
- **Discos compactos regrabables, CD-RW**(*CD-ReWritable*). Son similares a los CD-R, pero la capa de tinte está formada por una aleación de plata, indio, antimonio y telurio, que posee dos estados estables: cristalino y amorfo, con dos índices de reflexión distintos (alto y bajo, respectivamente). El láser actúa con tres potencias posibles:
  - ***alta:*** funde la aleación, convirtiéndola de estado cristalino a amorfo, representando una transición pozo-valle o valle-pozo;
  - ***media:*** funde la aleación, haciéndola pasar al estado cristalino de alta reflexión;
  - ***baja:*** no altera el estado de la aleación, y se utiliza para leer.

10.3.4.2 Disco Digital Versátil (DVD)

El fundamento físico de los DVD es el mismo que los CD, Existen también versiones DVD-R, DVD-RW (similares a las de los CD), con las siguientes mejoras:

- Los pozos son más pequeños (0,4 mm en lugar de 0,8 mm).
- La espiral es más pequeña (0,74 mm en lugar de 1,6 mm).
- La longitud de onda del láser menor (0,65 mm en lugar de 0,78 mm).
- Se pueden superponer dos capas, la primera de ellas semitransparente. El láser se enfoca a una capa u otra, dependiendo de la capa donde se desea que se refleje (se desee leer) (Figura 10.21).
- Se pueden grabar (y leer) ambas superficies.

Las tres primeras características hacen que se pueda grabar con una mayor densidad, y por tanto en la misma superficie se puede almacenar mayor información. También la utilización de distintas capas hace posible un incremento adicional de capacidad, existiendo unidades con la posibilidad de grabar hasta cuatro superficies (Figura 10.21).

Las unidades de DVD-ROM disponen de una caché usualmente de 256 ó 512 KB, donde almacenan 128 ó 256 sectores, respectivamente.

Los DVD pueden clasificarse, de acuerdo con su contenido, en los siguientes tipos:

- DVD-Vídeo, que almacenan películas (vídeo y audio).
- DVD-Audio, que almacenan música o voz en alta fidelidad.
- DVD-Datos, que contienen datos.

Según la forma de grabación se tienen las siguientes alternativas:

- DVD-ROM, sólo lectura grabado en fábrica en grandes tiradas (distribución de programas o datos).
- DVD-R y DVD+R, grabables por el usuario, pero una sola vez. Hay dos estándares, que se diferencian en si tienen un signo más (+) o menos (–) después de las siglas DVD. Cuando se adquiere un dispositivo DVD es conveniente fijarse es su compatibilidad. Si el dispositivo es DVD±R quiere decir que es compatible con ambos formatos.
- DVD-RW y DVD+RW, regrabables por el usuario.
- DVD-RAM, permite un acceso aleatorio a los datos, lo que le confiere una buena velocidad. Puede regrabarse más de 100.000 veces y usarse como disco duro adicional. Actualización de datos en la forma *lectura después de escritura* con la que inmediatamente después de escribir un dato se lee para comprobar que se ha grabado correctamente.

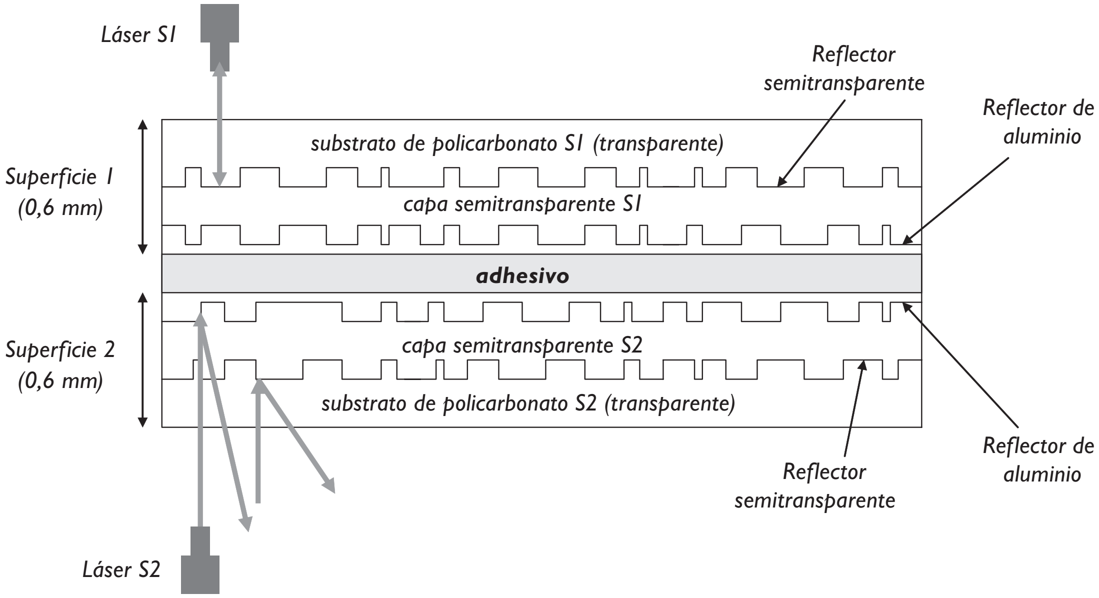


Figura 10.21. *Esquema simplificado de la estructura de un DVD.*

El disco puede tener una o las dos superficies grabadas, y una o dos capas de datos por cada superficie (Figura 10.21); el número de lados y capas determina la capacidad del disco, según se indica en la Tabla 10.6.

Tabla 10.6. *Distintas capacidades de discos DVD.*

	DVD de 8 cm de diámetro		DVD de 12 cm de diámetro	
1 supercicie, 1 capa	DVD-1	1,36 GB	DVD-5	4,38 GB
2 superficie, 1 capa	DVD-2	2,48 GB	DVD-9	7,95 GB
1 superficie, 2 capa	DVD-3	2,72 GB	DVD-10	8,75 GB
2 superficie, 2 capa	DVD-4	4,95 GB	DVD-18	15,9 GB

Las capas de grabación pueden ser determinadas visualmente por el número de lados de datos, y viendo cada lado. Las capas dobles son usualmente de color dorado, mientras que las capas simples son plateadas, como un CD. Otra manera de saber si un DVD contiene una o dos capas es ver el anillo central del disco. Si hay dos códigos de barras, es una capa doble. Si hay un código de barras, es una capa simple.

En la Tabla 10.7 se incluyen algunas características de discos ópticos. En los primeros CD la velocidad de transferencia era de 150 KB/s, y esta velocidad pasó a denominarse **factor de velocidad**. La velocidad 32X, quiere decir $150 \cdot 32 = 4{,}69$ MB/s. En los DVD el factor de velocidad es de 1,35 MB/s.

Tabla 10.7. *Parámetros usuales de distintos medios ópticos.*

Característica	CD	DVD (12cm)	CD-MO
Capacidad	650 MB	4,7 GB	300 MB a 5,2 GB
Factor de velocidad (x)	150 KB/s	1,35 MB/s	--
Velocidad de transferencia	12x a 52x	1 a 24x	5,92 MB/s[1]
Tiempo medio de acceso	80 a 150 ms	80 a 220 ms	28 ms[1]

[1] Datos de la Unidad Fujitsu 1300SD.

10.3.5 Discos magnetoópticos

Las unidades de **discos magnetoópticos** son dispositivos con los que es posible leer y escribir. La información, en vez de estar grabada físicamente de forma mecánica (por medio de un relieve de hoyos y valles, o por tintes), está grabada magnéticamente.

El fundamento físico de estos dispositivos es distinto al de las unidades de disco magnético y discos CD. El material donde se graba la información es una capa en la superficie del plato del disco formada por una aleación de terbio férrico y cobalto. Los discos vírgenes contienen una magnetización previa (todo a ceros, magnetización Norte, por ejemplo; Figura 10.22a). Cuando se desea grabar un disco (Figura 10.22b) toda su superficie se somete a un campo magnético débil y uniforme y contrario a la premagnetización del disco (Sur, por ejemplo), que no altera el estado de magnetización de cada celda individual. Un haz láser de unos 40 mW de potencia se enfoca con precisión sobre las celdas donde se desea escribir un 1. Al calentarse la celda cerca de su *punto de Curie* las partículas elementales de la aleación cambian de alineación de forma tal que se modifica su polaridad magnética (a Sur), grabándose así un 1.

La lectura se fundamenta en el *efecto Kerr*; según el cual el plano de polarización de un haz de luz gira en un sentido u otro según el estado de magnetización de la superficie en que se refleja (Figura 10.22c). La lectura del disco se efectúa con ayuda de un láser de unos 10 mW de potencia, captándose el haz de luz reflejada por un fotodetector después de haber pasado él mismo por un analizador óptico. Para regrabar el disco, primero se pone todo a ceros, siguiendo el mismo procedimiento anterior, pero borrando previamente los unos, aplicando el campo magnético externo con polaridad opuesta (N).

Tanto los CD-RW como los CD-MO son regrabables; aunque estos últimos son mucho más duraderos que los CD-RW. Los CD-RW se van degradando en cada operación de escritura (admiten entre medio millón y un millón de ciclos de borrado); sin embargo los CD-MO no se degradan cuando se re-escriben.

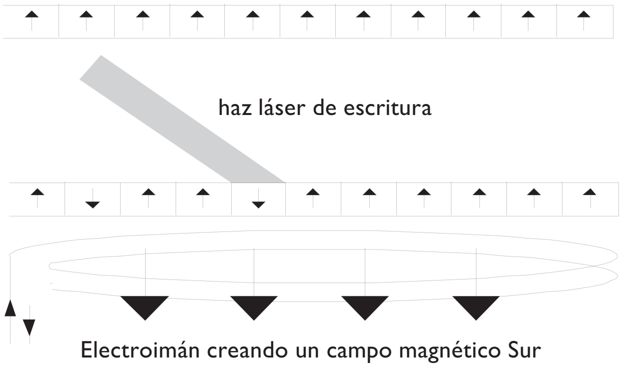


Figura 10.22. *Disco magnetoóptico (WMRA); (a) grabación original; (b) escritura;(c) lectura: el plano de polarización de la luz reflejada se desvía según sea la magnetización.*

En la última columna de la Tabla 10.7 se incluyen algunas características de una unidad de discos magnetoópticos.

EJEMPLO 10.13

Las especificaciones del disco magnetoóptico Fujitsu MCN3130AP son las siguientes:

- *Capacidad: 1,3 GB.*
- *Capacidad de 1 sector: 2 KB.*
- *Velocidad de transferencia maxima: 3,24 a 5,55 MB/s.*
- *Interfaz: ATAPI.*
- *Velocidad de transferencia de la interfaz ATAPI: 33,3 MB/s (UltraDMA mode2).*
- *Tiempo de búsqueda: 25 ms.*
- *Tiempo de espera (latencia): 10 ms.*
- *Velocidad de rotación: 4.000 rpm.*
- *Tiempo de carga: 13 segundos.*
- *Capacidad buffer: 1,8 MB.*

10.3.6 Memorias flash USB

Las memorias flash USB almacenan la información en una memoria semiconductora (uno o varios chips) de tipo flash NAND. Su cualidad más destacada es que, a pesar de ser memoria semiconductora, mantienen su contenido sin necesidad de suministrar energía eléctrica. En condiciones ideales el contenido de una memoria flash puede permanecer inalterado hasta 10 años, pudiendo ser regrabadas por más de un millón de veces. Se comportan como un disco magnético, y de hecho están sustituyendo rápidamente a los disquetes.

Las memorias semiconductoras flash son evolucionadas de las memorias EEPROM (chips de memorias programables y borrables eléctricamente). La diferencia esencial con las memorias RAM está en que cada celda (transistor) está formada por una pequeña capa de material conductor rodeada completamente de material aislante (*puerta flotante*), en la que con una

relativamente alta tensión se pueden inyectar electrones que quedan allí indefinidamente atrapados, manteniendo la información (cero o uno) de la celda, a pesar de que se desconecte la alimentación del circuito. Durante la grabación se cargan o descargan de electrones las puertas flotantes de cada celda (dependiendo de si se escribe un 0 o un 1), la operación de lectura es más sencilla y rápida, necesitándose niveles de tensión menores.

Las memorias flash USB utilizan un tipo de circuitos de memoria flash denominado NAND, en las que se accede a la información por bloques, siendo necesario para grabar uno de ellos previamente borrarlo (ponerlo a ceros) en su totalidad, y después escribir los unos donde deban estar. Los bloques de borrado suelen estar en el rango de 16 KB a 128 KB. El borrado, en comparación con las memorias EEPROM se realiza muy rápidamente, de hay el nombre de *flash* ("se borran en un flash").

Las memorias flash USB emulan el comportamiento de una unidad de disco, así:

- Un software denominado FTL (*Flash Translation Layer*) hace que el sistema operativo vea a la unidad flash como una unidad de disco.
- Se lee y escribe en bloques del tamaño de los sectores de un disco; es decir de 512 bytes; sin embargo, usualmente se lee o escribe en ráfagas de páginas de 2 KB que contienen cuatro bloques.
- El procesador central carga en los puertos del controlador la ubicación del fichero que se desea leer y su tamaño. El controlador se encarga de ir leyendo secuencialmente (por "sectores") el fichero e ir transfiriendo los datos a la interfaz USB.
- En la memoria flash se almacena un archivo que indica el contenido de la memoria en cuanto a directorios (carpetas) y archivos que tiene almacenados, así como la ubicación dentro de la memoria de los distintos archivos.

Los elementos básicos de una memoria flash USB son (Figura 10.23):

- Varios chips de memoria flash tipo NAND.
- Controlador, que dispone de un pequeño procesador para realizar la interfaz USB y la transformación de datos serie que recibe del computador en las páginas y bloques de datos requeridos por la memoria.
- Un oscilador de cristal que produce una señal de reloj de 12 MHz para sincronizar las operaciones internas y el intercambio de datos.
- Conector USB. La salida de la interfaz USB transmite la información en serie. Dispone de 4 cables, uno de alimentación, otro de tierra (o línea de referencia común), y dos, donde van los datos, en forma balanceada (si se transmite un 1 con 2,5 V, en una línea van +2,5V y en la otra –2,5 V). la alimentación obtenida a través de la conexión USB es de 5 voltios y de 100 a 500 mA.

Las características básicas son las siguientes:

- Volumen muy pequeño (les denominan *mecheros* o *lápices*), 30 gramos, y, por ejemplo, uno de 256 MB, contiene el equivalente a unos 177 disquetes.
- Capacidades: las primeras eran 8 MBytes, habiendo superado en la actualidad los 8 GBytes.
- Velocidad de lectura máxima alrededor de 100 Mbits/s; la de escritura un poco menor. La velocidad se suele dar aproximadamente con múltiplos de 150 KB/s; así, una memoria de velocidad de transferencia de 90x quiere decir que es de 90 · 150 KB/s = 13,18 MBytes/s.

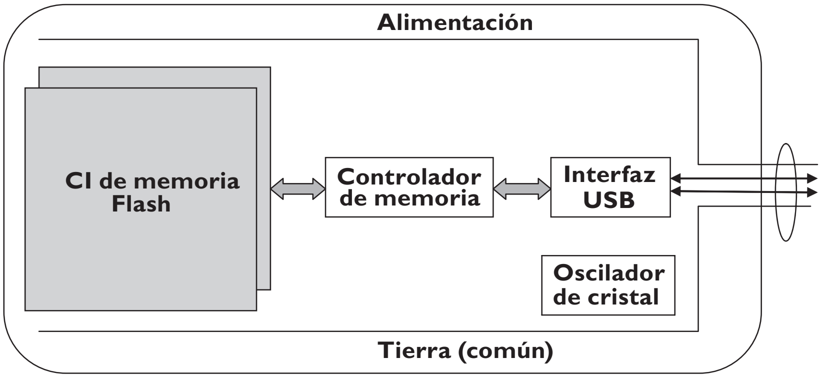


Figura 10.23. *Esquema simplificado de una memoria flash USB.*

En comparación con los discos magnéticos, como las memorias flash están realizadas con circuitos integrados, sin tener partes móviles, tiene tiempos de acceso y consumo mucho más bajos, y son insensibles a las vibraciones, lo que los hace muy útiles para equipos portátiles o móviles que se alimentan por pilas o baterías, tales como computadores móviles, cámaras digitales, reproductores de audio MP3, teléfonos móviles, etc.

10.4 Conclusión

En este capítulo, en primer lugar (Sección 10.1), hemos considerado el sistema de memoria globalmente y mostrando cómo existe una auténtica jerarquía entre los distintos tipos de ella, habiendo un compromiso entre velocidad y capacidad. De aquí surge la abstracción de jerarquía de memoria que justifica el apoyo del funcionamiento de los más rápidos en los más capaces, aunque más lentos, de su nivel inmediato inferior.

En segundo lugar nos centramos en la descripción de la memoria interna, describiendo la estructura y funcionamiento de la memoria caché (Sección 10.2.1) como medio para mejorar notablemente los tiempos de acceso del procesador al sistema de memoria, y detallando algunos aspectos de de la memoria principal (Sección 10.2.2) fundamentalmente en lo que se refiere a los tipos de circuitos integrados y módulos con las que se configuran y las formas de direccionamiento y acceso.

En cuanto a la memoria externa, último aspecto que considera el capítulo, se han revisado los sistemas que tienen una mayor proyección: discos magnéticos (Sección 10.3.2), cintas magnéticas (Sección 10.3.3), discos ópticos (Sección 10.3.4), discos magnetoópticos (Sección 10.3.5) y memorias flash USB (Sección 10.3.6).

EJERCICIOS

10.1 Un procesador dispone, entre otros, de los siguientes elementos: registro de dirección (AR) de 32 bits, registro de datos (DR) de 16 bits, contador de programa (PC), y puntero de pila (SP). Indicar:
1) Número de hilos del bus de datos.
2) Número de hilos del bus de direcciones.
3) Tamaño máximo posible de la memoria principal (en MB o GB).
4) Tamaño en bits del registro PC.
5) Tamaño en bits del registro SP.

10.2 Una memoria tiene una capacidad máxima de 64 Mpalabras de 32 bits y con acceso por

bytes. ¿Con cuántos bits se formarán las direcciones de memoria?

10.3 El bus de direcciones de una memoria con acceso por bytes contiene 32 líneas, y sus palabras son de 64 bits:
a) ¿Cuál es su capacidad máxima en palabras?
b) ¿Cuál es la dirección que debe darse a la memoria para acceder a la palabra cuya dirección es 13B7 54C7?
c) El byte de dirección CDA7 325A, ¿a qué palabra corresponde?

10.4 Estimar el ancho de banda que podría obtenerse con un sistema de memoria entrelazado con 4 módulos, sabiendo que cada módulo está organizado en palabras de 32 bits, y que el tiempo de ciclo de las memorias utilizadas es de 60 ns.

10.5 Suponga un sistema de memoria que consume 8 ciclos de reloj para acceder a una palabra, y 4 ciclos para accesos sucesivos y con frecuencia de reloj de 750 MHz. Si el procesador requiriese un bloque de 16 palabras, estimar el tiempo que se invertiría desde que el procesador solicita el bloque hasta que concluye su recepción, en los siguientes supuestos:
a) El acceso no se hace en forma de bloque, si no palabra a palabra.
b) El acceso se hace por bloques.
c) Se utiliza una memoria entrelazada de cuatro módulos.

10.6 Se desea diseñar un sistema de memoria que tenga un ancho de banda de $400 \cdot 10^6$ Bytes/s. Si se dispone de módulos de palabras de 16 bits con tiempos de ciclo de 15 ns, ¿Cuántos módulos habría que utilizar de forma entrelazada para conseguir el objetivo?

10.7 Suponga un sistema que no dispone de memoria virtual, y con tan sólo dos niveles de jerarquía (caché y memoria principal). Si los tiempos de acceso de los dispositivos utilizados son 5 ns y 50 ns, respectivamente, y el porcentaje de aciertos en caché del 95%. ¿Qué porcentaje de mejora se ha obtenido en el tiempo de ciclo al haber introducido la caché?

10.8 El sistema de memoria de un computador contiene una caché con tiempo de acceso de 4 ns, una memoria principal con tiempo de acceso de 80 ns, y una unidad de disco donde se gestiona la memoria virtual con tiempo de acceso de 12 ms. Después de analizar el comportamiento de la memoria se concluye que los porcentajes de aciertos en la caché son del 80% y en la memoria principal del 99,5%. Obtener los tiempos medios de acceso de caché y de memoria principal.

10.9 Se dispone de un sistema de memoria con una jerarquía de tres niveles: caché, memoria principal y memoria virtual. Se observa que de 1.500.000 referencias a memoria que hace el procesador, en 130.000 casos la dirección se encuentra en la caché y en 150.000 casos en la memoria principal. Obtener las tasas de acierto de los distintos niveles.

10.10 Una unidad de disco recibe la información del procesador en código ASCII (paridad impar), y utiliza como sistema de grabación magnética el MNRZI. Este sistema trata de lograr que nunca se graben más de dos ceros seguidos, consiguiéndose así mayores densidades de grabación, y un uno se representa por cambio de magnetización. Si la transformación de código es la que se indica en la Tabla 10.8, ¿cómo sería la información a grabar de los caracteres T7?

Tabla 10.8. *Transformación de código para grabación MNRZI.*

Grupo de 4 bits	5 bits grabados
0000	11001
0001	11011
0010	10010
0011	10011
0100	11101
0101	10101
0110	10110
0111	10111
1000	11010
1001	01001
1010	01010
1011	01011
1100	11110
1101	01101
1110	01110
1111	01111

10.11 Una unidad de paquetes de discos tiene 12 platos que giran a 7.200 rpm, con 24 cabezas de lectura/grabación, una por superficie. Si, una vez formateado el disco, cada superficie contiene 24.247 pistas con 793 sectores, por término medio, cada uno de 512 bytes, obtener:
a) El número de pistas por cilindro.
b) El número medio de bytes por pista.
c) La capacidad total de la unidad.

10.12 En la unidad de disco del ejercicio anterior, indicar la capacidad real del disco sin formatear, suponiendo que el formato que se le dio es el de la Figura 10.10. ¿Qué porcentaje de la unidad se desperdicia al dar el formato?

10.13 ¿Cuál debería ser el ancho de banda interno (entre la unidad física y la caché) de la unidad de los ejercicios anteriores suponiendo que en una revolución del disco se tuviese que leer una cuarta parte de los sectores de un cilindro?

10.14 Suponiendo que la unidad de disco del Ejercico 10.10 tiene un tiempo medio de búsqueda de la pista es $T_b=7,4\ ms$, y que, una vez que se accede a un sector se tardan $T_{lcs}=0,1\mu s$ en leer y comprobar el campo de identificación del sector, obtener el tiempo medio de acceso al campo de datos del sector.

10.15 Calcular la capacidad máxima de almacenamiento que puede soportar un controlador de disco IDE sabiendo que utiliza 10 bits para determinar el cilindro, 4 bits para el número de cabeza, y cada pista se divide en 63 sectores. ¿Cuál sería esta capacidad caso de utilizar una ROM-BIOS[4] mejorada que permite definir el número de cabeza con 8 bits, en lugar de con 4? (Suponer que en un sector se incluyen 512 Bytes de datos del usuario.)

10.16 En una unidad de discos RAID nivel 4, formada por 6 unidades de disco (D0 a D5) y que utilizada como redundancia un bit de paridad (criterio par), en un momento dado se produce una avería de la unidad D3. Suponiendo que la información de los 16 primeros bits de los discos es la que se indica en la Tabla 10.9.
a) Obtener los cambios que haría el sistema operativo para que el sistema siguiese funcionando correctamente, y los valores de los 16 primeros bits de los discos.

Cinco horas después la unidad se repara, y las unidades de disco, en el momento de volver a poder utilizar el disco D3, contienen la información que se indica en la Tabla 10.10.
b) Obtener los nuevos cambios, y los 16 bits iniciales de los discos que obtendría automáticamente el sistema

10.17 Una unidad de cinta opera a 800 bpi, con 9 pistas. Los bloques tienen el siguiente formato:
a) N Bytes de datos.
b) Zona en blanco equivalente a tres líneas, seguida de un carácter de comprobación de redundancias cíclicas (carácter CRCC).

Tabla 10.9. *Situación de las unidades de disco en el momento de la avería.*

DO	D1	D2	D3	D4	D5
0000 1010	1001 1100	0110 1101	averiado	0011 1111	1111 0010
1100 0101	1010 1011	1010 1101		1010 1001	1000 0111
……..	………	……..		……..	……..
……..	………	……..		……..	……..

Tabla 10.10. *Situación de las unidades de disco en el momento de reparar la avería.*

DO	D1	D2	D3	D4	D5
1100 1010	1000 1011	0011 0100	reparado	0110 1100	0110 1000
0101 1001	0000 0001	0010 1101		0100 0001	0101 1101
……..	………	……..		……..	……..
……..	………	……..		……..	……..

[4] Una ROM-BIOS es una ROM que contienen los PC compatibles con programas y parámetros de control de los periféricos estándar de entrada salida.

c) Zona en blanco de final de bloque. Ocupa el equivalente a 3 líneas. Esta zona también se denomina "blanco EOR". Es seguida de un carácter LPCC (carácter de comprobación de paridad longitudinal).
d) Interbloque (IRG), con L_{IRG} = 0,6".

Siendo la longitud del bloque de datos N = 512 B, ¿cuántos bloques y Bytes de datos se podrán almacenar en una cinta de 2.400 pies?

10.18 La separación de ficheros en la unidad de cinta descrita en el ejercicio anterior se realiza según la siguiente estructura:

1º Zona en blanco de final de fichero de 3" (EOF GAP).
2º Marca de Fin de Fichero (EOF): 1 Byte.
3º EOR GAP (blanco de final de registro): tres líneas en blanco.
4º Carácter LPCC (para comprobar paridad longitudinal).
5º IRG, con L_{IRG}=0,6".

¿Cuántos ficheros de 64 KBytes cabrán en una cinta de 2.400 pies grabada en la unidad indicada?

10.19 Una unidad de cinta utiliza el sistema de grabación PE, y opera a 1.600 bpi, con 9 pistas y v_t=75 ips. Los bloques tienen la siguiente estructura:

- 41 Bytes delimitadores del comienzo del bloque (40 todos 0, y 1 todo unos).
- N Bytes de datos (especificados por la longitud del bloque de datos).

41 Bytes delimitadores del final del bloque (1 todo 1, 40 todos 0). Siendo la longitud del interbloque L_{IRG} = 0,6" y N = 1.024, ¿cuántos bloques y Bytes de datos se podrán almacenar en una cinta de 2.400 pies?

10.20 La unidad del ejercicio anterior separa los ficheros con marcas *fin de fichero* también denominadas marcas EOF (*End of File*), y utilizando el siguiente formato:

…	………………………..
5º	41 Bytes delimitadores del final del último bloque del fichero M
1º	IRG, con L_{IRG}=0,6"
2º	Zona de 3" en blanco, de fin de fichero (análoga a un IRG)
3º	40 Bytes con marcas EOF
4º	IRG, con L_{IRG}=0,6"
5º	41 Bytes delimitadores del comienzo del 1er bloque del fichero M+1

Calcular la longitud de cinta que ocuparía un fichero de C = 64 KBytes de capacidad.

10.21 Estimar la capacidad máxima y velocidad de transferencia de datos, de una cinta de 2400 pies grabada a 1600 b/i, y que es leída por una unidad que funciona a una velocidad lineal de 75 i/s.

10.22 Calcular el número de bits de canal que se leen por segundo y la velocidad de transferencia media de datos de usuario, en una unidad de CD-ROM estándar, sabiendo que la velocidad de lectura es de 75 sectores por segundo. ¿Qué capacidad de datos cabría en un CD-ROM de 74 minutos?

10.23 Una unidad CD-ROM estándar tiene una velocidad de lectura de 75 sectores/segundo, y una capacidad para almacenar 74 minutos de tiempo. Calcular la longitud total del canal (pistas) de un CD-ROM. (*Recuérdese* que 1 sector contiene 2.352 Bytes, de los cuales 2 KB son de datos del usuario, y que la densidad de grabación de las pistas es de 0,6 μm/bit de canal).

BIBLIOGRAFÍA

[Bee05] Beekman, G.: *Introducción a la Informática;* 6ª Edc. Prentice Hall, 2005.

[Bro03] Brookshear, J. R.: *Computer Science*; 7th Edt. Addison Wesley, 20003.

[Coo95] Cook, B. M.; White, N. H.: *Computer Peripherals,* Edward Arnold, 3ª Edc. 1995.

[Doy99] Doyle, L. F.: *Computer Peripherals,* 2ª Edc. Prentice Hall, 1999.

[For03] Forouzan, B. A.: *Introducción a la Ciencia de la Computación,* Thomson, 2003.

[Ham03] Hamacher, C.; Vranesic, Z.; Zaky, S.: *Organización de computadores*, McGraw-Hill, 2003.

[Hob00] Hobbs, M.: *Multifuncional peripherals for PCs,* Newnes, 2000.

[Hol93] Holzbaur, H.; Hurd, J.: *32 high-speed hard drives*, Byte, pág.176-193, September 1993.

[Lio90] Lion, K.: *DAT's a Solution*, Byte, págs. 323-328. November 1990.

[Mee95] Mee, C. D.; Daniel, E. D. (Editores): *Magnetic recording technology,* 2ª Edc., McGraw-Hill, 1995.

[Mig04] Miguel, P. de: *Fundamentos de los computadores*, Thomsnon, 2004

[Min97] Minasi, M.: *The complete PC upgrade & maintenance guide,* 8ª Edc. Sybex, 1997.

[Mur00] Murdocca, M. J.; Heuring,V.P.: *Principles of Computer Architecture,* Prentice Hall, 2000.

[Nor03] Norton, P.: *Introduction to Computers,* 5th Edt. MacGraw-Hill, 2003.

[Ort05] Ortega, J.; Anguita, M.; Prieto, A.: *Arquitectura de Computadores*, Thomson, 2005.

[Pri05] Prieto, A.; Prieto, B.: *Conceptos de Informática,* Serie Shaum, McGraw-Hill, 2005.

[Ros99] Rosch, W. L.: *Hardware Bible,* 5ª Edc. Que, 1999.

[Sch97] Schmidt, F.: *The SCSI bus and IDE interface,* 2ª Edc. Addison Wesley, 1997.

[Sie90] Sierra, H. M.: *An introduction to direct access storage devices,* Academic Press, 1990.

[Sta00] Stallings, W.: *Organización y arquitectura de computadores*. 7ª Edc. Prentice Hall, 2006.

[Rao93] Raouf, F.: *First loocks. PCI: off to a flying start*, PC Magazine, pags.28-35, October 1993.

[Tan00] Tanenbaum, A. S.: *Organización de computadoras. Un enfoque estructurado,* 4ª Edc. Pearson Educación, 2000.

[Wil80] Wilkinson, B.; Horrocks, D.: *Computer Peripherals,* Hodder and Stoughton. 1980.

[Woo92] Woodard, O. C.; Long, T.: *Display Technologies,* Byte, págs.159-168, Julio 1992.

[Xer84] XEROX. *The Xerox 2700 Distributed Electronic Printer. System Management Guide,* 600P86/65. Septiembre 83. (Welwyn Hall). Rev. Enero. 1984.

CAPÍTULO 11

Periféricos de entrada/salida

En este capítulo se estudian los dispositivos (periféricos de entrada y salida) a través de los cuales el computador intercambia información con su mundo exterior Como es bien conocido existen muy diversas formas de dar información a un computador, como puede ser a través del lenguaje oral (reconocedores del habla), seleccionando iconos u opciones de un menú (ratón, pantallas sensibles al tacto y lápices), o por medio del lenguaje escrito (teclado). Por otra parte, existen dispositivos de salida inversos a los de entrada, que transforman la información binaria interna del computador en textos o imágenes o sonidos producidos por medio de pantallas, impresoras o altavoces. El presente capítulo pretende dar una visión general acerca de ellos.

En primer lugar (Sección 11.1) se define el término periférico y se enumeran las funciones principales de estos dispositivos, se hace una clasificación de los mismos y se incluyen algunas características generales que son comunes a todos ellos y que son de gran utilidad.

Posteriormente se describen los periféricos tradicionales de entrada (Sección 11.2) y de salida (Sección 11.3) de mayor interés. Finalmente se dedican dos secciones específicas a los periféricos para aplicaciones multimedia (Sección 11.4) y a entradas/salidas para señales analógicas.

El estudio detallado de cada periférico va más allá de las pretensiones de esta obra, por lo que se hace una descripción somera de los mismos, aunque a causa de su interés pedagógico, se hace hincapié en alguno de ellos.

11.1 Definición, objetivos y tipos de periféricos

Se denominan **periféricos** tanto a las unidades o dispositivos a través de los cuales el procesador se comunica con el mundo exterior, como a los sistemas que almacenan o archivan información, sirviendo de memoria auxiliar de la memoria principal (**memoria externa**). Es decir, un periférico es cualquier dispositivo del computador que no sean sus unidades centrales (el procesador o la memoria principal). El mundo exterior con el que intercambia información las unidades centrales del computador puede ser: una persona, dispositivos de memoria masiva, otro computador o, en la mayoría de los sistemas embebidos, un sistema a controlar (Tabla 11.1).

La memoria externa ha sido estudiada en el capítulo anterior (Capítulo 10). Recordemos que trata de suplir las deficiencias de la memoria central en cuanto a su relativamente baja capacidad y el hecho de que la información almacenada en la zona RAM se borra al desconectar la alimentación de energía eléctrica (al desconectarla).

Tabla 11.1. *Clasificación de periféricos según los sistemas externos a comunicar con las unidades centrales del computador.*

Comunicación persona-computador:	Periféricos de entrada.
Comunicación computador-persona:	Periféricos de salida.
Comunicación computador central-memoria externa, y viceversa:	Periféricos de memoria auxiliar (disco magnético, cinta magnética, CD, DVD, etc.). *Estudiados en el Capítulo 10.*
Comunicación computador-computador:	Módem, tarjetas o circuitos de red (Ethernet, WiFi, Bluetooth, etc.). *Estudiados en el Capítulo 18*.
Comunicación para supervisión y control de otros sistemas:	Conversores A/D y D/A, etc.

Según la definición de periférico dada anteriormente, éstos están constituidos por unidades de entrada, unidades de salida y unidades de memoria externa. Estas últimas unidades también pueden considerarse como unidades mixtas (tanto de entrada como de salida), ya que el procesador y la memoria principal pueden escribir (dar salidas) en ellas, y la información escrita puede ser leída; es decir, actuar como entrada. Ahora bien, la información grabada en estos soportes no es directamente inteligible para el usuario del computador (que constituye el principal elemento *exterior* al mismo); esto es, no puede haber una intercomunicación directa usuario-computador como la que hay, por ejemplo, a través de un teclado/pantalla.

El computador es una máquina que no tendría sentido si no se comunicase con el exterior, es decir, si careciese de periféricos de E/S. Debe disponer de:

- unidades de entrada, a través de las cuales le podamos dar los programas que queramos que ejecute, y los datos correspondientes, y de
- unidades de salida, con las que el computador nos da los resultados de los programas.

Los **dispositivos de E/S** transforman la información externa en señales eléctricas codificadas permitiendo su transmisión, detección, interpretación, procesamiento y almacenamiento de forma automática. Los dispositivos de entrada convierten la información de entrada (instrucciones o datos tecleados, por ejemplo) según alguno de los códigos (ASCII, Unicode, etc.) vistos en la Sección 4.2. Así el procesador y la memoria reciben dicha información adecuadamente preparada (en binario). En un dispositivo de salida (impresora, por ejemplo) se efectúa el proceso inverso: la información binaria que llega del procesador o de la memoria (caracteres representados en código ASCII, por ejemplo) se transforma de acuerdo con el código de E/S en caracteres escritos inteligibles para el usuario.

No hay que confundir *periférico* con *soporte* de información. Es distinto una unidad DVD (sistema para grabar o leer), o una impresora, que sus soportes de información: un disco DVD o un listado de impresora, respectivamente.

Cada periférico suele estar formado por dos partes claramente diferenciadas en cuanto a su misión y funcionamiento: una parte mecánica y otra parte electrónica.

Los *elementos mecánicos* están constituidos básicamente por dispositivos electromecánicos (conmutadores manuales, relés, motores, electroimanes, servomecanismos, etc.), controlados por los elementos electrónicos. La velocidad de funcionamiento de un periférico y el tiempo medio transcurrido entre averías suelen venir impuestos por los elementos mecánicos.

Los *elementos electrónicos* o **controlador del periférico** se encargan de interpretar las órdenes que le llegan del procesador para la recepción o emisión de datos, dependiendo de que se trate de un periférico de salida o entrada, respectivamente, y de generar las señales

de control para la activación de los elementos electromecánicos del periférico que producen o captan los datos en el soporte de información correspondiente (pantalla, impresora, disco magnético, etc.). En la parte electrónica de los periféricos es común usar *elementos optoelectrónicos* que actúan como detectores o generadores de la información de entrada o salida, respectivamente. También estos dispositivos se utilizan como *detectores de posición* de los elementos mecánicos móviles del periférico.

En el capítulo anterior (Capítulo 10) analizamos los dispositivos de memoria masiva (memoria externa) y en el presente capítulo se pretende describir y analizar el funcionamiento de los periféricos de Entrada/Salida de mayor interés de los anteriormente relacionados.

En las tablas 11.2 y 11.3 se enumeran los periféricos más usuales, la mayoría de los cuales describiremos en las secciones siguientes. Existen otras unidades (Tabla 11.3) que actúan conjuntamente como de entrada y salida. Los **módems** (**moduladores/demoduladores**) y tarjetas de red que sirven para transmitir datos a través de líneas de teléfono, se estudiarán en el Capítulo 18, dedicado a transmisión de datos.

Tabla 11.2. *Ejemplos de unidades de entrada.*

• Lectora de tarjetas perforada[1]	
• Lectora de cinta de papel perforada[1]	
• Teclado	
• Detectores ópticos	– de marcas
	– de barras impresas
	– de caracteres impresos
	– de caracteres manuscritos
	– Escáner de imágenes
	– Cámara de fotos
	– Cámara de vídeo
• Tarjeta de edición de vídeo	
• Lectora de banda magnética	
• Detector de caracteres magnetizables	
• Sensores-conversores analógico/digital (Interfaz industrial de entrada)	
• Unidad de reconocimiento de la voz	
• Lápices óptico, electrostático y de presión	
• Pantalla sensible al tacto	
• Palanca manual de control (*joy-stick*)	
• Digitalizador o tableta gráfica	
• Ratón	
• Cámara fotográfica digital	

[1] Dispositivo obsoleto.

Tabla 11.3. *Ejemplos de unidades de salida.*

Unidad perforadora de tarjetas[1]	
Unidad perforadora de cinta de papel[1]	
Tarjeta de edición de vídeo	
Monitores de visualización	– Pantallas de rayos catódicos (CRT) – Pantallas planas
Impresora	
Conversor digital/analógico-efector (Interfaz industrial de salida)	
Sintetizador de voz	
Visualizadores (*displays*)	
Registrador gráfico (*plotter*)	

[1] Dispositivo obsoleto.

Tabla 11.4. *Ejemplo de unidades mixtas (entrada y salida).*

• Terminal interactivo teclado/pantalla[1]
• Terminal teletipo[1]
• Pantalla sensible al tacto
• MÓDEM

[1] Dispositivo obsoleto.

Los primeros computadores utilizaban como principales unidades de entrada (además de los interruptores del panel de control), cintas de papel perforadas, y posteriormente (hasta la década de los años 70) tarjetas de cartulina (17,85 x 7,68 cm). En ambas la información se codificada mediante perforaciones (o ausencias de ellas) en determinadas posiciones. También esta técnica se utilizaba en unidades de salida, para obtener resultados utilizables como entrada de otros programas. En el capítulo vamos a describir brevemente otros dispositivos de entrada y salida más actuales, y haremos especial hincapié en los de mayor uso, indicando sus peculiaridades.

11.2 Periféricos clásicos de entrada

En esta sección, en primer lugar (Sección 11.2.1), se describe el teclado, como uno de los dispositivos de entrada más comunes. A continuación (Sección 11.2.2) se analizan brevemente sistemas de entrada directa manual como son los paneles sensibles al tacto, los lápices ópticos, la palanca para juegos y el ratón. Un gran número de dispositivos de entrada están basados en el uso de detectores ópticos, y a ellos les dedicamos la Sección 11.2.3. Para completar el panorama de periféricos de entrada, en la Sección 11.2.4 tratamos de los dispositivos de captura directa de datos magnetizables.

11.2.1 Teclado

Los teclados son similares a los de una máquina de escribir, correspondiendo cada tecla a uno o a varios caracteres, funciones u órdenes. Para seleccionar uno de los caracteres de una

tecla, al igual que ocurre en una máquina de escribir convencional con los números y con las mayúsculas y minúsculas, puede ser necesario pulsar simultáneamente dos o más teclas, una de ellas la tecla correspondiente al carácter.

Al pulsar una tecla se cierra un conmutador que hay en el interior del teclado, esto hace que unos circuitos codificadores del **controlador del teclado** generen el código correspondiente al carácter seleccionado (ASCII, por ejemplo), almacenándolo en la memoria intermedia del teclado. El controlador del teclado envía una petición de interrupción al procesador para que, cuando sea aceptada la interrupción, el programa **gestor del teclado** (Sección 13.7) capte el código llevándolo de la memoria intermedia (*buffer*) al procesador, donde permanece hasta que algún programa lo requiera en una operación de entrada. Normalmente el programa gestor del teclado, hace un *eco* del carácter pulsado, visualizándolo en la pantalla de salida que se suele asociar al teclado.

En la Figura 11.1 puede verse un esquema que muestra la disposición usual de las teclas en un teclado.

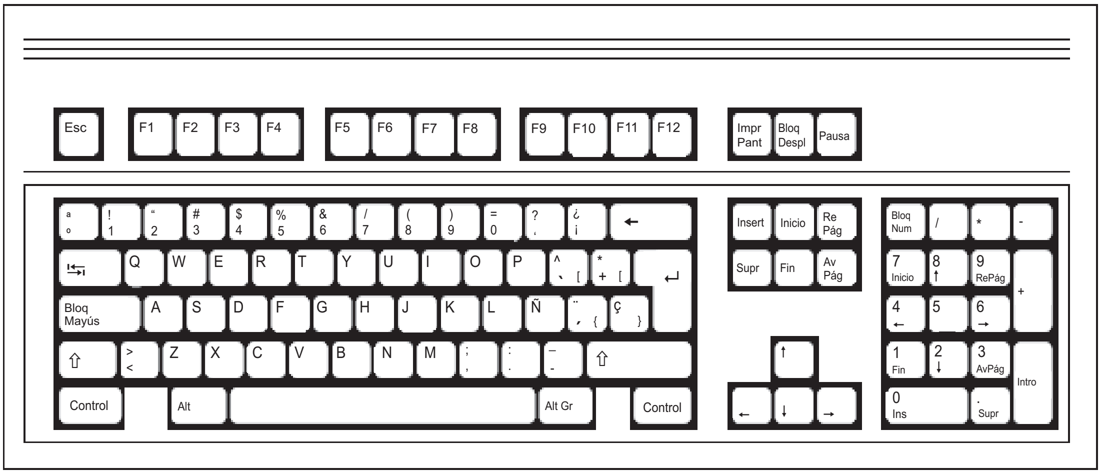


Figura 11.1. *Esquema simplificado de la distribución de las teclas en un teclado convencional.*

Según las normas ANSI (*American National Standars Institute*) los teclados deben contener los siguientes tipos de teclas:

- **Teclado principal.** Contiene los caracteres alfabéticos, numéricos y especiales, como en una máquina de escribir convencional con alguno adicional.
- **Teclas de gestión de imagen.** Sobre la pantalla se visualiza una marca o **cursor** o **indicador de posición**, que puede tener la forma de un cuadrado, o un ángulo, u otra marca, que suele visualizarse intermitente o muy brillante, destacando sobre el resto de los símbolos representados en la imagen. El cursor indica la posición de la pantalla donde aparecerá el siguiente carácter que tecleemos o la posición donde está el control del terminal. Las teclas de gestión de imagen permiten realizar operaciones tales como repetir el último carácter seleccionado, desplazar el cursor a izquierda, derecha, arriba y abajo (teclas con flecha en la figura), borrar un carácter o parte de una línea, etc.

Esto, entre otras cosas, posibilita que podamos modificar el contenido de la memoria intermedia de salida antes de enviar la información al procesador (antes de pulsar la tecla "ENTER").

- **Teclado numérico.** Es habitual en los teclados de computadores que las teclas correspondientes a los caracteres numéricos (cifras decimales), signos de operaciones básicas (+, –, etc.), y punto decimal estén repetidas para facilitar al usuario la introducción de datos numéricos. El teclado de la Figura 11.1 contiene 14 teclas de este tipo.
- **Teclas de funciones.** (12, en la Figura 11.1). Son teclas cuyas funciones son definibles por el usuario o están predefinidas en un paquete de programas o aplicación dada. Al pulsar una de estas teclas (F1, ..., F12) se transmite una secuencia de caracteres que, por ejemplo, puede dar lugar a la ejecución de una determinada parte o rutina de la aplicación en ejecución. Supongamos, por ejemplo, que el usuario está utilizando un programa de gestión bancaria. El programador de esta aplicación podría haber asociado la tecla F1 a "actualización de cuenta corriente", la F2 a "obtención de saldo", la F3 a "ingreso de dinero", la F4 a "reintegro de dinero", etc. El operador del terminal no tendría más que pulsar una de estas teclas para que efectúe la acción requerida.
- **Teclas de funciones locales.** Controlan funciones propias del terminal de que forme parte el teclado, como forma de cursor a aparecer en la pantalla, velocidad del desplazamiento vertical de imagen cuando el procesador está enviando información, impresión por la impresora acoplada al terminal del contenido de imagen, tecla de escape, tecla de avance de página, tecla para ir al inicio de un documento, etc.

Existen dos tipos de teclados: mecánicos y de membrana.

- En un **teclado mecánico** cada tecla es un pulsador que al ser oprimido provoca un contacto en una terminación metálica del circuito impreso del propio teclado, cerrando así un circuito específico del carácter. Al cesar la opresión de la tecla, un muelle la levanta llevándola a su posición inicial. Un procesador del teclado identifica el carácter seleccionado y genera e introduce en el *buffer* el código correspondiente.
- En los **teclados de membrana** se utilizan tres capas, que de arriba a abajo son las siguientes:
  - capa que contiene los dibujos de los caracteres y en la parte inferior unas placas conductoras;
  - capa intermedia de goma con agujeros justo debajo de cada una de las teclas;
  - capa conductora con pequeñas protuberancias coincidiendo con los agujeros de la capa de goma.

  Al pulsar una tecla, se oprime la goma entrando en contacto la plaquita conductora con la protuberancia de la capa inferior, cerrándose así un circuito eléctrico, cuya corriente y posición es detectada por el procesador del teclado. Estos teclados son más económicos que los mecánicos, pero su calidad de funcionamiento y tiempo de vida son menores, al deformarse la goma con el uso.

En el Problema 9.10 del Capítulo 9 se describe el funcionamiento interno de un teclado. El teclado, junto con el ratón, es uno de los dispositivos de entrada más utilizados, y se han desarrollado **teclados ergonómicos** (algunos de ellos divididos en dos partes, para cada una de las manos), que tratan de adaptarse más a la anatomía y movimiento cómodo de las manos y dedos.

11.2.2 Entradas manuales directas

Como alternativa a los teclados existen diversos dispositivos que se adaptan a la mano del usuario y permiten seleccionar directamente opciones de menú, iconos o dibujar objetos o escribir directamente en el periférico. Este es el caso del ratón, la pantalla sensible al tacto, los lápices y la palanca de control para juegos, dispositivos que se describen brevemente en las siguientes secciones.

11.2.2.1 Pantallas y paneles sensibles al tacto

Son pantallas o paneles que pueden detectar las coordenadas (*x, y*) de la zona de la propia pantalla donde se aplica una presión (por ejemplo, con un dedo).

La pantalla puede contener embebida una red rectangular o matriz de hilos conductores muy finos, paralelos a los ejes *x* e *y* de la imagen. Al presionar se hace que entren en contacto determinados hilos: uno *x* con otro *y*. El contacto eléctrico producido puede detectarse fácilmente, identificándose así la posición del punto presionado. Otras pantallas de menor resolución funcionan mediante retículas de células fotoeléctricas. Las de mayor resolución utilizan una gran cantidad de diminutas fibras ópticas integradas en su interior. En definitiva, estos dispositivos se fundamentan en la localización de un punto de una superficie donde se ha producido un cambio en sus propiedades eléctricas u ópticas.

Existen sistemas compactos, que llevan integrado el sistema de salida (pantalla de visualización) y el de entrada (detección), y externas, en las que se acopla un módulo sensible al tacto (con aspecto de un filtro de pantalla) en la parte delantera de un monitor convencional, y su salida se lleva, por ejemplo, al puerto serie.

Lo usual es producir sobre la pantalla (desde un programa) un menú de opciones, con zonas especificadas asociadas a cada una de ellas, donde debe presionar el usuario. La opción se selecciona sin más que tocar la zona correspondiente.

Las pantallas sensibles al tacto constituyen un sistema muy sencillo e intuitivo para dar entradas o elegir opciones sin utilizar el teclado. Puede ser útil para usuarios de computadores principiantes, tales como niños de corta edad. También es corriente utilizarlas para cajas registradoras, terminales punto de venta[1], gasolineras, etc. Este tipo de pantallas también se utiliza en las PDA y en sitios donde el equipamiento informático debe ser lo más compacto y robusto posible, para evitar averías y deterioro por las condiciones ambientales. Los paneles o almohadillas táctiles (*touchpad*) se utilizan en equipos portátiles; y, combinadas con pequeñas pantallas de cristal líquido, para operaciones de entrada y salida en agendas personales (PDA).

11.2.2.2 Lápices óptico, electrostático y de presión

Un **lápiz óptico** físicamente (Figura 11.2) tiene la forma de una pluma o lápiz grueso, de uno de cuyos extremos sale un cable con un conector para unirlo a un monitor de pantalla o a una tableta especial. En el otro extremo hay una abertura o ventana por la que puede pasar la radiación luminosa de la pantalla hacia un sensor óptico que la detecta. Esta radiación es enfocada internamente o llevada por una fibra óptica a la superficie de un fotodetector. La señal eléctrica generada en él, después de ser amplificada, se transmite a los circuitos

[1] Un **terminal punto (o puesto) de ventas** o **terminal POS** (***P****oint*-***O****f*-***S****ale*) es una caja registradora de un centro comercial que en realidad es un terminal interactivo de una red local, y que dispone de lector de barras impresas, lector de tarjetas magnéticas, etc., de forma que el control de existencias, de ventas y de caja se hace de forma centralizada.

de control del monitor. El lápiz contiene un pulsador, transmitiéndose información hacia el monitor sólo en el caso de estar presionado o estar pulsado un conmutador que puede contener en un lateral.

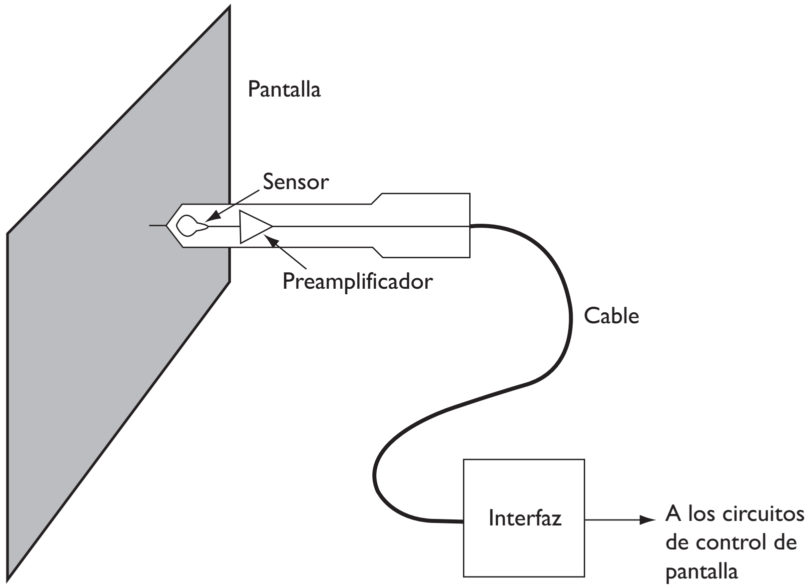


Figura 11.2. *Esquema simplificado de un lápiz óptico.*

En una pantalla CRT (Sección 11.3.2.1) la imagen se forma por medio de un haz de electrones que barre internamente la pantalla, línea a línea, con una frecuencia de 0,02 segundos. Si desde el programa en ejecución se activa el lápiz óptico, al colocar la ventana de éste frente a un punto (x_0,y_0) de la pantalla, en el instante t_0 de incidir el haz de electrones en el punto señalado, el sensor óptico capta la luz generada y produce un pulso eléctrico. A partir del valor t_0 los circuitos de control pueden obtener las coordenadas (x_0,y_0) donde estaba el haz de electrones en $t = t_0$, o lo que es lo mismo, la posición exacta del lápiz en la pantalla. El programa en curso podrá ejecutar un bloque de instrucciones o subrutina en función de las coordenadas del punto marcado por el usuario (el programa obtiene x_0,y_0).

Además de los lápices ópticos existen otros de los siguientes tipos:

- **Lápiz electrostático**: genera descargas electrostáticas que son detectadas de forma similar a la indicada, pero la superficie sensible de la pantalla, en lugar de contener detectores de luz, contiene detectores electrostáticos.
- **Lápiz de presión**: se utiliza en conjunción con paneles sensibles al tacto, similares a los vistos en la sección anterior.

Los lápices se utilizan frecuentemente para dar entradas o seleccionar opciones de menús en los computadores móviles (PDA, por ejemplo), estando la interfaz contenida en el controlador de vídeo.

11.2.2.3 Ratones

Un **ratón** es un dispositivo de entrada que sirve para introducir información gráfica o seleccionar coordenadas (x,y) de una pantalla. Los usos más habituales del ratón se encuentran en

la selección de iconos u opciones de menús, selección y arrastre de ventanas y otros objetos, y dibujo manual de líneas u otros elementos geométricos. Al desplazar el ratón manualmente sobre una alfombrilla o superficie plana, se reproducen los movimientos en el cursor de la pantalla. Por lo general, cada vez que se desplaza el ratón una distancia mínima (por ejemplo, 0.01") se envía (en forma serie) una secuencia compuesta de 3 Bytes, indicando cada uno de ellos: desplazamiento X, desplazamiento Y, y estado de los pulsadores. Los ratones detectan, por tanto, movimientos relativos. Cuando se desea seleccionar un icono u opción de un menú, en primer lugar, desplazando adecuadamente el ratón, debe posicionarse el cursor encima del objeto a seleccionar y después realizar una o dos pulsaciones (clics) en uno de los dos pulsadores que lleva el ratón. El ratón puede considerarse como uno de los dispositivos de entrada más populares y para muchas funciones más cómodo que un teclado.

Existen tres tipos básicos de ratones:

- **Ratón mecánico.** Está internamente constituido por una bola que puede girar libremente, y se acciona haciéndola rodar sobre una superficie. La bola es solidaria con dos rodamientos o sensores perpendiculares entre sí, cuyos desplazamientos se detectan eléctricamente en forma similar a los de una palanca para juegos (Figura 11.3). Actualmente está en desuso.
- **Ratón ópto-mecánico.** En este caso la bola hace girar dos ruedas, o dentadas o con pequeñas ventanas regularmente dispuestas en la parte más externa, situadas perpendicularmente una a otra. Cada rueda dentada se encuentra entre un diodo emisor de luz (LED) y un fotodetector, de forma tal que el desplazamiento (ángulo de giro) de la rueda se puede detectar con los impulsos de luz que recibe el detector (cuando en el camino óptico no se interponen los dientes de la rueda o pasa a través de una de las ventanas). Los valores recibidos de los dos fotodetectores son llevados a sendos contadores binarios (X e Y), que se incrementan o decrementan conforme sea el sentido, horizontal o vertical, del movimiento de la bola del ratón. Esta información es transmitida por unos cables flexibles ("cola del ratón") al computador, y el programa gestor del ratón puede determinar la distancia, dirección y sentido del desplazamiento, desde que se inició el último movimiento.
- **Ratón óptico.** Contiene un emisor de luz y un detector de la luz reflejada. El ratón se hace mover sobre una base que tiene dibujada una retícula de dimensiones prefijadas. La detección de los movimientos relativos se efectúa midiendo la intensidad de luz reflejada que variará al pasar el fotodetector por encima de las líneas de la retícula. Otros ratones más modernos no necesitan almohadilla con retícula pudiéndose utilizar sobre cualquier superficie; disponen de una pequeña retina artificial que capta unas 1.500 imágenes por segundo, comparando los cambios que se producen en imágenes sucesivas se determinan con precisión los desplazamientos realizados. Los ratones ópticos, por tanto, no tienen bola ni elementos móviles por lo que tienen una mayor fiabilidad y vida, y además tienen mayor precisión que los mecánicos.

Con los computadores portátiles o en situaciones en las que no hay espacio suficiente para desplazar el ratón, suelen utilizarse **ratones estacionarios** (*trackball*). Son como ratones mecánicos pero que se utilizan con la bola hacia arriba, de forma que ésta se desplaza con el dedo pulgar, y no haciéndola rodar por una superficie. De hecho hay ratones que pueden utilizarse en las dos formas indicadas.

Tradicionalmente un ratón se solía conectar al computador a través de un conector PS/2 o de un puerto serie RS-232, pero en la actualidad es común hacerlo a través de una interfaz

USB (Capítulo 12). También hay ratones inalámbricos que emiten por radio la información serie que genera. Esta información es captada por un pequeño emisor/receptor conectado a una interfaz USB.

EJEMPLO 11.1

Un ratón[2] óptico inalámbrico puede tener las siguientes características:

- *Base USB o PS/2 que conecta por radio frecuencia (RF) con el ratón en un radio de acción de 2 metros.*
- *Resolución del sensor óptico: 800 dpi.*
- *Velocidad máxima de captura: 4,7 Megapíxeles/s.*
- *Funcionamiento correcto hasta una aceleración de: 10 g.*
- *Funcionamiento correcto hasta una velocidad de: 1 m/s.*

11.2.2.4 Palancas para juegos (joystick)

Una **palanca manual de control** o mando para juegos (en inglés: *joystick*) está constituida por una cajita en cuya superficie externa se insertan diversos conmutadores y pulsadores, y de la que sale verticalmente una palanca o mando móvil. El usuario puede actuar sobre el extremo de la palanca exterior a la caja, y a cada posición de ella le corresponde un punto de tres coordenadas (*x,y,z*). La cajita o varilla dispone de un pulsador que debe ser presionado para que exista una interacción entre el programa y la posición de la varilla.

Con las instrucciones adecuadas el programa puede captar el estado del pulsador (presionado o no) y las coordenadas (*x,y,z*) correspondientes a la posición de la varilla. De esta manera puede lograrse que el cursor, o una figura o motivo, se desplacen tridimensionalmente en la pantalla siguiendo las posiciones correspondientes al movimiento de la palanca. Esta función es muy útil en videojuegos y aplicaciones gráficas.

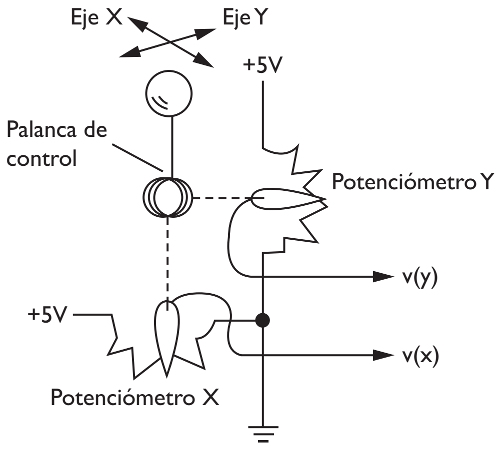


Figura 11.3. *Esquema eléctrico del interior de una palanca de control.*

[2] Logitech® MX™700.

La cajita contiene en su interior dos potenciómetros circulares y perpendiculares, situados en la superficie interior de un casquete esférico, cuyos ejes móviles están unidos solidariamente a la palanca de control. En la Figura 11.3 se esquematiza el circuito; con él se obtienen en todo momento tres tensiones, *v(x)* , *v(y)* y *v(z)*, proporcionales a los valores de x, y, z respectivamente. Estos valores analógicos pueden ser transformados en digitales por un conversor A/D. Otros modelos contienen diversos microinterruptores distribuidos perpendicularmente en cuatro direcciones, que se cierran en función de la posición de la palanca.

En definitiva, una palanca para juegos, a diferencia de un ratón, permite transferir movimiento a un objeto no sólo en dos sino en las tres dimensiones del espacio.

11.2.3. Detectores ópticos

Hay ciertos documentos o productos que se utilizan en la vida ordinaria en gran cantidad y que pueden ser leídos por computador, introduciendo con rapidez y sin error sus características sin necesidad de teclear el código o información que los identifica. Esto es así porque en su superficie (o envoltorio) llevan impresos caracteres, barras o marcas predefinidos, que pueden ser detectados por sistemas especiales, normalmente de tipo optoelectrónico o magnético que se suelen denominar **dispositivos de captura directa de datos**.

Como ejemplos de los productos y documentos citados anteriormente están los talones o cheques bancarios, productos comerciales, quinielas, impresos para corrección de exámenes tipo test de elección múltiple, etc.

En la mayoría de estos sistemas existe un conjunto de caracteres o formas (**patrones**) predefinidos. Los datos de identificación se imprimen con los patrones establecidos. Los lectores, por lo general, analizan los datos carácter a carácter, descomponiendo cada uno de ellos según una retícula o mosaico compuesto de celdillas o zonas de identificación. Se detecta si cada zona de identificación está impresa (situación que se representa, por ejemplo, con 1 lógico) o no (0 lógico). A cada carácter o forma, por tanto, se le hace corresponder una secuencia ordenada de ceros y unos. El dispositivo de entrada compara esta secuencia con la de los patrones (que tiene grabadas internamente), identificando la forma en cuestión. En caso de no identificarse el dato, se procede a una nueva lectura y se da una señal de error para que el operario de la unidad de lectura conozca esta eventualidad.

Además de los dispositivos ópticos citados anteriormente, existen otros denominados **escáneres**, que se limitan a captar y digitalizar imágenes grabadas en papel, representándolas internamente tal y como se explicó en la Sección 4.4, sin realizar ningún reconocimiento de formas, cosa que se puede realizar posteriormente con programas especiales. A estos dispositivos les dedicaremos la Sección 11.2.3.4.

Los lectores ópticos suelen contener una fuente de luz que ilumina intensamente el dato a leer, un sistema óptico de ampliación de la imagen, y los elementos necesarios para *barrer* el carácter o descomponer la imagen en las celdillas o zonas de identificación de patrones. También incluyen los circuitos electrónicos de identificación, en su caso, y de transmisión del código binario correspondiente al bus del computador.

La presentación del documento frente a la unidad de detección óptica se hace siguiendo uno de los tres sistemas que indicamos a continuación. El primero de ellos consiste en que los documentos a leer se ubican en un cajetín o depósito de alimentación y por un sistema mecánico o neumático de arrastre son llevados uno a uno a través de una ranura frente al elemento de lectura, pasando posteriormente al cajetín de salida. Este es el sistema utilizado

para talones bancarios y escrutinio de quinielas, por ejemplo. El segundo sistema, no tan automatizado, consiste en introducir y sacar manualmente uno a uno los documentos en la ranura donde está instalada la unidad de detección. En el último sistema, el menos automatizado y, por tanto, más manual, la lectura se efectúa haciendo pasar el sistema detector (que tiene forma de lápiz o de pistola) frente a la zona de la etiqueta o documento donde se encuentran las marcas o caracteres a leer.

En las siguientes secciones se describirán brevemente los detectores ópticos citados anteriormente.

11.2.3.1 Detectores de marcas

Los lectores ópticos de *marcas* u **OMR** (*Optical Mark Reader*) son sistemas que aceptan información escrita a mano y la transforman en datos binarios inteligibles por el computador. El usuario se limita a marcar con su lápiz o pluma ciertas áreas o posiciones preestablecidas del documento, que representan posibles opciones de determinados eventos o preguntas. Estos documentos pueden ser leídos posteriormente, a gran velocidad, por el computador con un lector óptico de marcas. Éste detecta las zonas preestablecidas que están marcadas.

Esta forma de introducir datos en el computador es útil, por ejemplo, para corregir exámenes de elección múltiple, escrutar quinielas, valorar encuestas, o en una fábrica donde la ejecución de determinadas tareas de producción se registra marcando a lápiz determinados rectángulos impresos en una ficha.

11.2.3.2 Detectores de códigos de barras

Han adquirido un gran desarrollo los lectores de códigos de barras, que se usan con mucha frecuencia en grandes almacenes, supermercados y comercio, en general para identificar productos, llevar un control exacto de inventario, controlar las operaciones de carga y descarga de mercancías, agilizar las ventas, etc. En el momento de fabricar un producto, se imprime en su envoltorio una etiqueta con información sobre el mismo según un código formado por un conjunto de barras separadas por zonas en blanco. La información de estas etiquetas puede ser captada rápida y fácilmente por detectores de códigos de barras para su posterior procesamiento.

El **código de barras** es la representación de una determinada información mediante un conjunto de líneas paralelas verticales de diferente grosor y espaciado (Figura 11.4). La correspondencia entre los datos que representan y el código de barras se denomina *simbología*. Las simbologías pueden ser clasificadas en dos grupos, atendiendo a dos criterios diferentes:

- Continuo o discreto: según que los caracteres en las simbologías continuas, comienzan con un espacio y el siguiente comienza con una barra, o viceversa. Los caracteres en las simbologías discretas comienzan y terminan con barras; el espacio entre caracteres se hace muy estrecho de forma que se diferencie claramente del final del código de un carácter.
- Dos grosores o múltiples grosores: en el primer caso las barras son anchas o estrechas, no teniendo significado su grosor, que además puede variar de un carácter al siguiente. En las simbologías de múltiples grosores la anchura de las barras son múltiplos de una predeterminada, X, denominada **módulo**. Generalmente, se emplean barras con anchura *X, 2X, 3X* y *4X*.

Figura 11.4. *Código de barras de productos alimenticios de empresas españolas.*

Las simbologías descritas se denominan lineales. Hay también simbologías de dos dimensiones (2D), con una gran variedad de formatos, un ejemplo de los cuales se encuentra en la Figura 11.5. El más común consiste en una matriz de código formada por la inserción de formas cuadradas o de puntos en una retícula bidimensional.

Figura 11.5. *Ejemplo de información representada por un código de barras 2D.*

EJEMPLO 11.2

Según el código EAN-13 cada producto se marca con 13 dígitos, en el orden y con el significado que se da a continuación (Figura 11.4).

- *Prefijo: 2 dígitos. Código del estado donde se ha inscrito la empresa al sistema EAN producto. Todas las empresas adscritas al sistema de la Asociación Española de Codificación Comercial (AECOC) codifican sus artículos con el número 84 como prefijo. Esto no indica que el producto haya sido fabricado en España sino que la empresa, independientemente de su nacionalidad y de su ubicación, utiliza el sistema de código de barras asignado por AECOC.*
- *Código del propietario de la marca: de 5 a 8 dígitos: Está compuesto por el prefijo y un número, de entre 5 y 8 dígitos. Este código no identifica al fabricante del producto, sino al propietario de la marca.*
- *Código de identificación del producto: de 5 a 2 dígitos.*
- *Verificación o autocomprobación de error: 1 dígito.*

La suma de los dígitos de empresa y producto tiene que ser 10. El dígito de control se obtiene sumando los dígitos de las posiciones impares, se multiplica por 3, se le suman los dígitos de las posiciones pares y a este resultado se le resta el siguiente múltiplo de 10.

Existen varios códigos normalizados, siendo el más utilizado en el mundo el código lineal EAN (*European Article Numbering*), ya que ha sido adoptado por más de 100 países. En la actualidad este código de barras es gestionado por la organización mundial GS1, con sede en Bélgica. La forma de codificar cada dígito decimal consiste en variar el grosor relativo de las barras negras y blancas adyacentes.

Con las marcas o etiquetas con código de barras se puede, por ejemplo, controlar fácilmente por computador las existencias y ventas de una determinada empresa. Cuando un cliente va a abonar en caja, el dependiente pasa una lectora óptica por la etiqueta (o viceversa), introduciéndose así, sin necesidad de teclear, y con rapidez, la identificación del artículo. El computador contabiliza el producto como vendido y lo da de baja en la base de datos de existencias. El lector óptico suele formar parte de una caja registradora que en realidad es un terminal interactivo especializado denominado **terminal punto (o puesto) de ventas** o **terminal POS** ("**P**oint-**O**f-**S**ale").

EJEMPLO 11.3

En el código EAN, el grosor de las barras negras y espacios blancos es simple, doble o triple de un cierto valor (0,33 mm) denominado módulo. Para cada dígito se reservan 7 módulos (es decir, 2,31 mm) donde debe haber dos barras y dos espacios (cada espacio es un conjunto de módulos consecutivos en blanco). Adicionalmente se incluyen tres separadores: dos laterales que delimitan el principio y fin del código y se componen de sólo tres módulos (negro, blanco y negro), y uno central que separa los códigos de producto y de fabricante y se compone de 5 módulos (blanco, negro, blanco, negro, blanco). Se suelen utilizar simultáneamente tres juegos (A, B, y C) de códigos. El que para un dígito concreto se utilice uno u otro juego depende de su posición dentro del código global, o del primer dígito. En la Tabla 11.5 se representa el código EAN, cada bit representa un módulo, siendo el 1 un módulo negro y el cero un módulo blanco. Las barras de los extremos son especiales y distintas, de forma que el lector puede reorientar automáticamente la etiqueta para lograr una lectura correcta.

Tabla 11.5. *Códigos de barra EAN. (Cada bit representa un módulo de barra de 2,31 mm, 1 negro y 0 blanco).*

	Juego A	Juego B	Juego C
0	0001101	0100111	1110010
1	0011001	0110011	1100110
2	0010011	0011011	1101100
3	0111101	0100001	1000010
4	0100011	0011101	1011100
5	0110001	0111001	1001110
6	0101111	0000101	1000100
7	0111011	0010001	1000100
8	0110111	0001001	1001000
9	0001011	0010111	1110100

Para leer códigos de barra se suele utilizar un escáner portátil con baterías propias (autónomo) conectado a un dispositivo de memoria externa, tal como una memoria flash USB o un disco duro USB, aunque también se puede transmitir la información captada por el escáner de forma inalámbrica a través de una interfaz WiFi o Bluetooth. Para la lectura de códigos 2D se suele utilizar una cámara normal, decodificándose la información por software.

11.2.3.3 Digitalizadores

Los **digitalizadores** o **tabletas digitalizadoras** son unidades de entrada que permiten transferir directamente al computador o editar con gran precisión gráficas, figuras, planos, mapas, fotos o dibujos en general. También pueden ser utilizadas por dibujantes o diseñadores gráficos para introducir en el computador dibujos o figuras realizados a mano alzada. Esto se hace (Figura 11.6) pasando manualmente una pieza móvil (lápiz o cursor) por encima de la línea a digitalizar (como si se estuviese *calcando*), automáticamente se transfieren las coordenadas (x,y) de los distintos puntos que forman la imagen, unas detrás de otras. En cierta forma, por tanto, un digitalizador realiza la función contraria a la de registrador gráfico: partiendo de un dibujo se obtiene una representación digital de él, en el interior del computador.

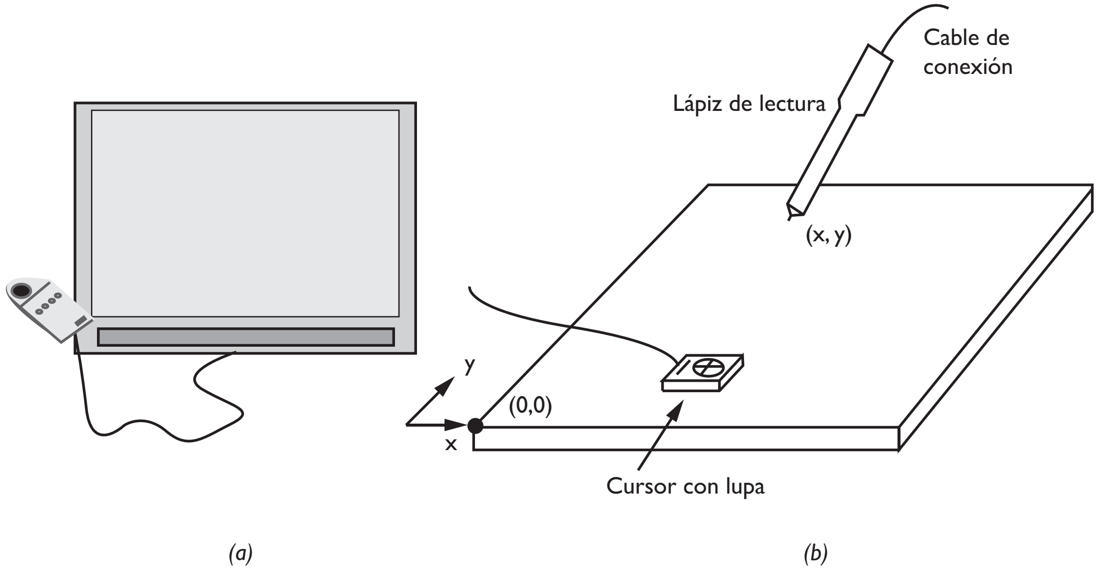


Figura 11.6. *Mesa o tabla digitalizadora: (a) esquema mostrando los dos tipos posibles de mando (lápiz o cursor con lupa); (b) aspecto general de una tableta digitalizadora.*

Todo digitalizador consta de tres elementos:

- **Tabla o tablero rectangular,** que puede ser opaco o transparente, donde se ubica el dibujo a digitalizar.
- **Mando** con el que el operador debe recorrer el dibujo. Éste puede tener forma de lápiz o de cajoncito (del tamaño aproximado de un mando a distancia de un televisor), y está unido al resto del sistema por un cable flexible. En el último caso el mando tiene una ventana cerrada con una lupa, en cuyo interior se encuentra embebida una retícula en forma de cruz; ésta sirve para señalar o apuntar con precisión el punto a digitalizar. El mando puede disponer de uno o varios pulsadores para controlar la modalidad de funcionamiento, forma de transmisión, y selección de opciones o rutinas del programa que gestiona la digitalización.
- **Circuitos electrónicos**. Controlan el funcionamiento de la unidad. Usualmente la detección de coordenadas es de tipo analógico, siendo la señal obtenida amplificada y convertida a digital por los circuitos del digitalizador, antes de ser transmitida al computador.

Además de las rutinas del sistema operativo encargadas de controlar la transmisión de información (como ocurre en todo periférico) se suele disponer de rutinas especializadas en el tratamiento de los datos generados por el digitalizador. Estas rutinas pueden realizar funciones tales como:

- Obtención de la distancia entre dos puntos.
- Obtención de los ángulos de una recta determinada y los ejes (*x,y*) del tablero.
- Cálculo del perímetro y área de una figura cerrada, previamente digitalizada, etc.

Los digitalizadores, junto con los escáneres, registradores gráficos y pantallas gráficas con ratón, son elementos fundamentales de los **sistemas gráficos**, que tienen en la actualidad gran importancia en diversas aplicaciones de la Informática (véase Sección 1.8). Entre estas áreas de aplicación se encuentra el **diseño con ayuda de computador (CAD)**: arquitectura e ingeniería, diseño y confección textil, cartografía, diseño de circuitos impresos e integrados, dibujo artístico, etc.

11.2.3.4 Escáneres de imágenes

Un escáner de imágenes es un sistema para digitalización de documentos en papel, basado en la exploración (*scanning*) de imágenes mediante procedimientos optoelectrónicos; transforma la información contendida en una página (documento, fotografía, imágenes fijas, etc.) en una señal eléctrica que, con la interfaz adecuada, es transmitida a un computador convencional o, a través de línea de teléfono o conexión inalámbrica, a una impresora lejana (caso de un equipo de fax), o procesada dentro de un fotocopiadora para mejorar la calidad de la misma, etc.

El sistema considera a una página (documento, fotografía, etc.) como dividida en una fina retícula de celdas o puntos de imagen, que son iluminados por una fuente de luz. Esta luz se refleja en cada celda, y una malla de sensores optoelectrónicos convierte la luz reflejada en una carga eléctrica, o sea, en una señal analógica. Las señales analógicas obtenidas como consecuencia del barrido de la página son digitalizadas por un conversor A/D (Sección 11.5), conformando así la imagen captada en un **mapa de bits**, almacenable y procesable en un computador (Sección 4.4).

Un escáner de imágenes contiene tres elementos funcionales básicos: un detector (con la electrónica asociada), una fuente de luz y lentes de barrido. La fuente de luz ilumina el objeto y las lentes forman la imagen del objeto en el detector. La elección de la fuente de iluminación y de las lentes depende de las características del detector. El documento debe ser iluminado con suficiente luz de forma que la luz reflejada y enfocada por las lentes sea suficiente para permitir al detector operar con una adecuada relación señal/ruido. A veces la fuente de luz es un haz generado por un diodo láser que barre los distintos puntos de la retícula de la imagen.

En la actualidad se suelen utilizar, dependiendo del modelo, tres tipos de circuitos integrados como detectores: **retículas de fotodiodos**, **retículas de dispositivos acoplados por carga (CCD)**, **retículas de fotodiodos acoplados por carga (CCPD)**. Las retículas CCD y CCPD son más fáciles de incluir en un escáner que las de fotodiodos. Las de CCD, además, pueden operar con niveles de iluminación muy bajos ya que detectan los fotones dentro de intervalos de tiempo grandes.

Existen dos técnicas básicas para capturar la imagen: plano-focal (*focal-plane*) y de rodillo (*platen-based*). La técnica de plano-focal usa una retícula bidimensional de detectores, focalizándose directamente la imagen del documento en ella. Este sistema es similar al de

las cámaras de TV, y no se obtiene calidad muy alta. La otra técnica, de rodillo, es similar a la utilizada en las fotocopiadoras. El documento se sitúa es una superficie de cristal, plana o cilíndrica, y la imagen se va captando conforme el detector se va desplazando delante del documento. El detector está constituido por una hilera de cientos de sensores por centímetro, cuya longitud total corresponde con la dimensión menor de la superficie máxima a digitalizar. Cuanto mayor sea el número de sensores, y más juntos se encuentren, mayor será la **resolución**, y mayor información se tendrá sobre la forma de la figura y, por tanto, de mayor calidad será la imagen captada. La calidad y precisión de los sensores determina los niveles de gris que el sistema es capaz de distinguir. Así, por ejemplo, algunos escáner pueden detectar 64 tonos de grises (el 0 codificaría el blanco y el 63 el negro, por ejemplo) mientras otros pueden detectar 256 niveles, o incluso más. Los niveles analógicos se convierten a digital por medio de un conversor A/D; es decir, se transforman en señales binarias.

Los escáneres de color suelen disponer de tres fuentes de luz, correspondientes a los tres colores básicos (rojo, verde y azul). Con la superposición de las tres imágenes que se forman iluminando con cada uno de las fuentes se puede almacenar y reconstruir la imagen en color (véase Sección 4.4).

EJEMPLO 11.4

Un escáner de color tiene una resolución de 1200 dpi y puede detectar 256 niveles para cada uno de los tres colores básicos. Estimar la memoria necesaria para almacenar una fotografía de 10 x 13 cm.

La fotografía tiene un total de:

$$\frac{10 \cdot 13}{2,54} = 51,18\ p^{\cdot}$$

En una pulgada cuadrada (p^2) hay:

$$1.200 \cdot 1.200 = 1.440.000 \text{ puntos de imagen}$$

Es decir, el número de puntos a digitalizar es:

$$51,18\ p \cdot 1.440.000\ \frac{puntos}{p} = 73.899.200\ puntos = 70,93\ Megapuntos$$

Para codificar cada color se necesitan 8 bits, ya que con 8 bits se pueden codificar 2^8=256 niveles. Como hay 3 colores se necesitan 24 bits, es decir 3 bytes, para codificar el color de cada punto de imagen. Por lo que la capacidad de memoria necesaria para almacenar la imagen será:

$$70,23\ Mpuntos \cdot 3\frac{Bytes}{punto} = 211\ MB$$

Recuérdese que existen técnicas especiales para comprensión de imágenes, tal y como se vio en la Sección 4.7.

La cantidad de información generada por un escáner es muy grande, por lo que para obtener un buen rendimiento de este periférico para transferir la imagen al computador se suelen utilizar interfaces paralelo y de alta velocidad (Sección 12.5). En la actualidad la interfaz más habitual es la USB que ha sustituido a la interfaz paralela. Otra interfaz que se usa en escáneres profesionales es el SCSI, que está siendo sustituida progresivamente por la FireWire.

Otro parámetro de interés en un escáner óptico es la velocidad de exploración, que suele darse en páginas DIN-A4/segundo.

Hay cuatro tipos básicos de escáneres: manuales (los hay incluso en forma de pluma), de sobremesa, de tambor, y de diapositivas. Los manuales son más difíciles de utilizar y su resolución y velocidad suele ser menor que los otros tres tipos. Los de sobremesa son más cómodos (detectan directamente páginas de tamaño DIN-A4 o A3, según los modelos), y los de diapositivas suelen ser los de mayor resolución. Los de alimentación continua son similares a los de sobremesa, pero llevan un sistema automático para transportar o alimentar los documentos a captar, situándolos debajo de la superficie de detección.

Un escáner se usa, como cualquier otro periférico, en conjunción con un programa gestor que forma parte del sistema operativo y que adapta las peculiaridades de la imagen captada a las del computador que va a utilizarla: resolución, paleta de colores, etc. Además, existen paquetes software especializados para gráficos y dibujos que permiten procesar las imágenes.

EJEMPLO 11.5

Un escáner[3] puede tener las siguientes características:

- *Resolución óptica: 2.400 dpi (seleccionable de 25 a 9.600 dpi).*
- *Interfaz: USB.*
- *Fuente de luz: LED (diodo emisor de luz).*
- *Profundidad de color: 48 bits.*
- *Escala de grises: 16 bits.*
- *Tamaño del documento: A4.*
- *Velocidad de vista preliminar (antes de escanear): 9 s.*
- *Consumo eléctrico: 2,5 W en funcionamiento, 1,5 W en espera.*

Según indicamos anteriormente, las imágenes digitalizadas generan un mapa de bits; pero puede utilizarse un programa de **reconocimiento óptico de caracteres** (**OCR**, *Optical Character Recognition*) para detectar e identificar los caracteres (alfabéticos y numéricos) impresos o mecanografiados. De esta forma, una imagen en mapa de bits se puede convertir en texto (un archivo ASCII, por ejemplo) posteriormente editable e imprimible por el usuario.

Los modelos OCR con menos errores de reconocimiento son los que utilizan patrones normalizados, tal como los de los estándares OCR-A (europeo) y OCR-B (Estados Unidos). En esencia lo que hace el programa OCR es identificar (por las zonas en blanco de separación entre los caracteres) las zonas de puntos que corresponden a un carácter, y posteriormente compara esta información con patrones que tiene almacenados de los distintos caracteres reconocibles. Una zona se identifica con el carácter-patrón que más se asemeje. Esta aplicación de los sistemas OCR es de gran interés, ya que permite, por ejemplo, introducir en un computador el contenido de un libro, sin necesidad de teclearlo, para posteriormente editarlo.

También hay reconocedores de caracteres escritos a mano que se denominan sistemas **ICR** (*Intelligent Carácter Recognition*), son menos fiables que los OCR, dado que existe una gran diversidad en la forma de escribir cada carácter concreto entre personas distintas e incluso en una misma persona, sirviendo frecuentemente sólo para detectar unos pocos caracteres (numéricos, por ejemplo). Usualmente en el manual del dispositivo se indica la caligrafía *preferida* por el periférico.

[3] CanoScan LiDE 80.

EJEMPLO 11.6

Algunos programas[4] *pueden utilizarse como OMR, decodificador de códigos de barras, OCR e ICR Algunas de sus posibilidades son las que se indican a continuación.*

Como **OMR:**

- *Reconoce marcas de texto sencillas hechas a mano con un porcentaje de precision del 99,995%.*

Como *decodificador de códigos de barra* es capaz de reconocer:

- *Códigos de barra 1D de más de 15 tipos distintos (entre otros los EAN).*
- *Capaz de reconocer códigos de barra 2D, del estándar PDF417, capaz de codificar hasta 1,1 KB de datos, incluyendo texto e imágenes.*

Como **OCR**, *capaz de reconocer:*

- *Hasta 188 lenguajes distintos escritos a máquina.*
- *Documentos multilíngües.*
- *Caracteres chinos, japoneses y coreanos (CJK).*
- *Caracteres OCR-A, OCR-B y MICR (E13B).*
- *Textos escritos entre 1600 y 1937 con caracteres especiales (letra gótica, etc), en ingles, francés, alemán, italiano y español*[5].

Como **ICR:**

- *Reconoce hasta un total de 91 lenguajes escritos a mano.*
- *Reconoce documentos multilengua.*
- *Admite 22 estilos de escritura a mano de diferentes países y áreas: Europa, América, Asia, Canadá Rusia, Japón, Arabia y Taití.*
- *Admite dígitos ICR hindúes, usados en los países árabes.*

Por otra parte, admite documentos de entrada en los siguientes formatos de entrada:

- *BMP*
- *PCX, DCX*
- *JPEG*
- *PNG: black and white, gray, color.*
- *TIFF*
- *PDF*
- *GIF*
- *DjVu*

Puede generar los textos en los siguientes formatos:

- *Microsoft Word (DOC).*
- *Rich Text Format (RTF).*
- *Microsoft Word XML (XML), Microsoft Office Word 2003 only.*
- *XML nativo.*
- *Adobe® Acrobat® (PDF).*
- *HTML.*
- *Microsoft PowerPoint (PPT).*
- *Comma Separated Values (CSV).*
- *Texto plano (TXT) y Unicode.*
- *Microsoft Excel (XLS).*
- *DBF.*

---

[4] Por ejemplo, el *ABBYY FineReader Engine 8.0.*

[5] Para ello puede utilizarse el módulo *FineReader XIX* del *ABBYY FineReader Engine 8.0.*

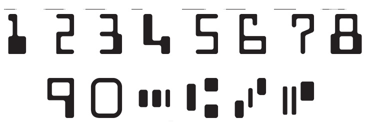
En la actualidad con frecuencia se usan sistemas que incorporan en una sola unidad impresora y escáner, denominándose **impresora multifuncional**.

11.2.4 Dispositivos de captura directa de datos magnetizados

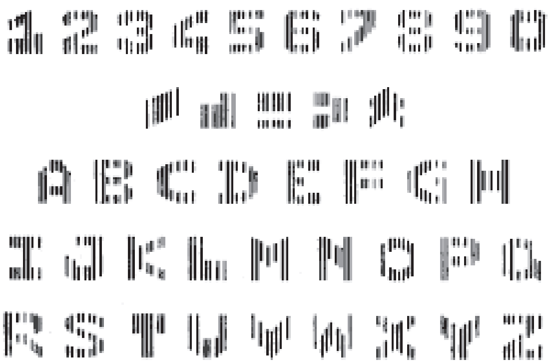
Otros dispositivos de captura directa de datos se basan en la medida de campos magnéticos, como los detectores de caracteres magnetizables y los detectores de bandas magnéticas, que se describen brevemente a continuación.

11.2.4.1 Detectores de caracteres magnetizables

Los caracteres magnetizables se utilizan, por ejemplo, en los talones y cheques bancarios, y en las etiquetas de los medicamentos. En estos documentos se imprimen, de acuerdo con unos patrones que se muestran en la Figura 11.7, los caracteres que identifican el cheque o talón. La tinta utilizada es magnetizable (contiene óxido de hierro) y además es legible directamente por el hombre. La impresión se hace con una máquina auxiliar denominada **inscriptora electrónica**. La lectora de caracteres magnéticos contiene una unidad de magnetización por la que pasan los documentos antes de llegar a la estación de lectura. El elemento de lectura contiene una microbobina que va barriendo el carácter e induciendo en ella un potencial eléctrico proporcional a la cantidad de tinta impresa bajo su radio de acción.



(a)



(b)

Figura 11.7. *Formas de caracteres magnetizables; (a) Normalización E13B; (b) Normalización CMC7.*

11.2.4.2 Lectores de bandas magnéticas

Las tarjetas de compra y de crédito, así como otras tarjetas de identificación, suelen llevar una banda de un material plástico recubierto de un material magnetizable, tal como óxido de hierro u óxido de cromo. En esta banda se graban los datos de identificación del usuario de la tarjeta, que pueden ser leídos por dispositivos especiales, tales como los que se encuentran en los cajeros bancarios o lectores de tarjetas de créditos.

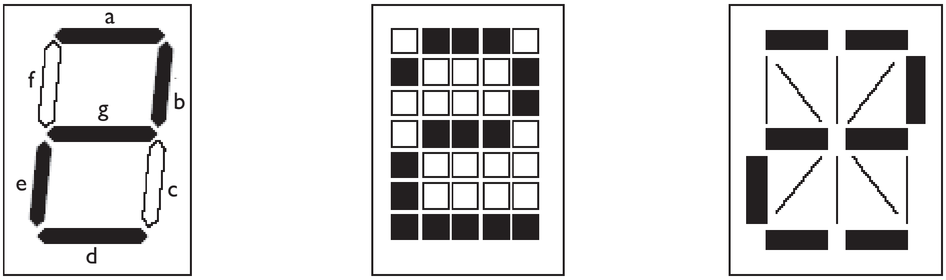
11.3 Periféricos clásicos de salida

En está sección se analizan periféricos de salida tradicionales como son: visualizadores (Sección 11.3.1), monitores (Sección 11.3.2), impresoras (Sección 11.3.3) y registrador gráfico (Sección 11.3.4).

11.3.1 Visualizadores

Los **visualizadores** (*displays*) son pequeñas unidades de salida que permiten al usuario leer información producida por el computador.

Los caracteres se forman partiendo de estructuras o módulos, cada uno de los cuales sirve para visualizar un carácter (numérico, alfabético, o especial).



(a) *(b)* *(c)*

Figura 11.8. *Módulos o indicadores para formar visualizadores. (a) de 7 segmentos; (b) de matriz 7 x 5; (c) de 16 segmentos.*

Cada módulo (Figura 11.8) contiene una serie de segmentos: 7 ó 16 ó 7 x 5 = 35, siendo los más habituales de 7. Un carácter concreto se visualiza activando determinados segmentos, dependiendo de la forma del carácter. Así, en el caso de la Figura 11.8a, para visualizar:

- un 2 hay que activar los segmentos a, b, g, y e;
- una F hay que activar los segmentos a, f, y g.

Es fácil realizar un circuito decodificador que transforme, por ejemplo, caracteres del código ASCII en las señales activadoras de los módulos de la Figura 11.8, para visualizar dichos caracteres.

El visualizador es el elemento de salida típico de las calculadoras de bolsillo, y suele estar constituido por módulos de 7 segmentos situados bajo una ventana protegida con un cristal o plástico transparente.

Corrientemente como segmentos se suelen utilizar diodos emisores de luz (LED) o de cristal líquido (LCD).

11.3.2 Monitores de visualización

Un **monitor** es un periférico de salida de un computador que permite visualizar la información en forma de imágenes y textos. Se conecta al computador a través de un controlador de vídeo también denominado adaptador o tarjeta gráfica o de vídeo. La forma más cómoda de adquirir información es a través de la vista, por lo que los monitores de visualización constituyen hoy día el sistema más cómodo y usual de captar las salidas de un computador.

Recordemos que según se indicó en la Sección 4.4, una imagen en un computador se representa por un conjunto discreto de puntos. De igual manera, la imagen de una pantalla no es continua, formándose en la retina del usuario por la yuxtaposición de multitud de **píxeles** (puntos de imagen). Por otra parte la imagen se forma físicamente en la pantalla del monitor por la activación selectiva de multitud de elementos denominados **puntos de pantalla** o **subpuntos** (*dot pitch*), que en las pantallas en que la iluminación se produce por excitación de pigmentos de fósforo se denominan **luminóforos**. Un punto de pantalla se iluminará más cuanto mayor sea la activación del elemento correspondiente. En las pantallas de color, el color de cada punto de imagen se obtiene con mezcla de tres colores básicos: rojo, verde y azul. Para ello en la pantalla se encuentran distribuidos alternativamente en todas las direcciones puntos de pantalla emisores de los tres colores (Figura 11.9). Como consecuencia de lo anterior un punto de imagen no puede ser menor que un punto de pantalla: en una pantalla monocroma podrán coincidir, pero en una pantalla en color un punto de imagen está formado al menos por tres puntos de pantalla. Un punto de imagen debe emitir toda una gama de colores, con lo que debe estar formado con al menos un punto de pantalla de cada uno de los colores básicos, activables por separado. Programando de forma individual la intensidad de cada color básico, el ojo humano detectará la mezcla obtenida en cada punto de imagen, pudiéndose producir así, por mezcla, una gran gama de sensaciones de colores distintos. En la Tabla 11.6 se indican los colores que se obtienen mezclando los tres colores básicos con igual intensidad.

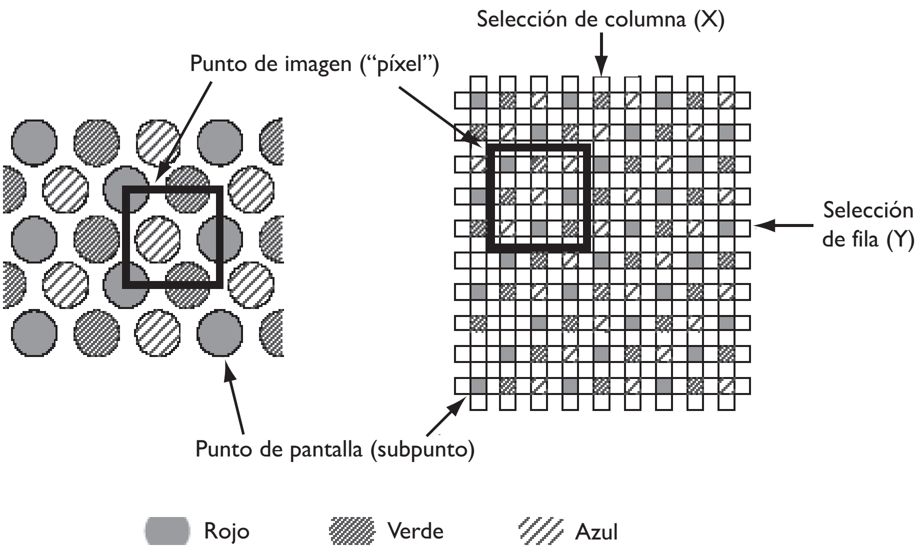


Figura 11.9. *En un monitor de color un píxel está formado por al menos tres puntos de pantalla.*

Cuando la pantalla se utiliza para visualizar textos, se considera dividida en **celdas**, en cada una de las cuales puede ir un carácter. La celda está constituida por una matriz regular de puntos de imagen. Así, por ejemplo, en una pantalla una celda puede contener 9 x 14 (alto x ancho) puntos de imagen formándose un carácter con 9 x 7 puntos. En la Figura 11.10 se representa una celda de una pantalla que está constituida por 12 x 10 puntos de imagen (12 filas y 10 columnas).

Tabla 11.6. *Obtención de distintos colores a partir de mezclas de igual intensidad de los tres colores básicos.*

RGB	COLOR
0 0 0	Negro
0 0 1	Azul
0 1 0	Verde
0 1 1	Azul celeste (Cian)
1 0 0	Rojo
1 0 1	Rosa oscuro (Magenta)
1 1 0	Amarillo
1 1 1	Blanco

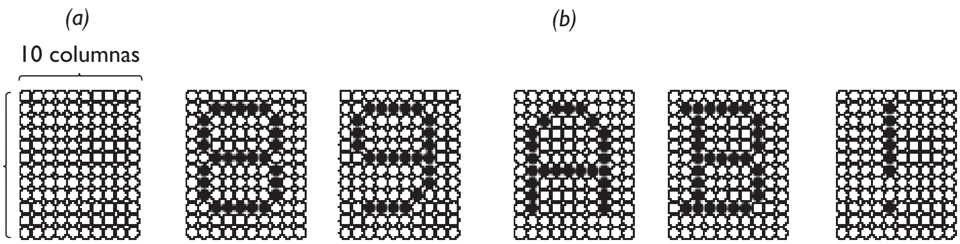


Figura 11.10. *(a) Puntos de imagen de una celda de un terminal Dasher D-460; (b) representación de algunos caracteres en el terminal citado.*

Tabla 11.7. *Ejemplos de resoluciones.*

Tipo de tarjeta	Resolución	Nº colores
MDA	720 x 350	b/n
HGC	720 x 350	b/n
CGA[1]	640 x 200	b/n
EGA	640 x 350	16
VGA[2]	640 x 480	16
SVGA	800 x 600	256
XGA	1.024 x 768	256
Macintosh	1.152 x 870	-
SXGA	1.280 x 1.024	2^{24}
UXGA	1.600 x 1.200	2^{24}
QXGA	2.048 x 1.536	2^{24}

[1] Se puede utilizar también con 4 colores, bajando la resolución a 320 x 200.
[2] Se puede utilizar también con 256 colores, bajando la resolución a 320 x 200.

Un monitor de visualización está constituido por dos elementos básicos: una **pantalla de vídeo** y un **controlador de vídeo** (o **controlador gráfico**). Estos elementos se analizarán en las secciones 11.3.2.1 a 11.3.2.3.

Teniendo en cuenta la información a visualizar hay dos tipos de monitores: monitores de caracteres (ASCII, por lo general), y monitores gráficos. Los **monitores de caracteres** actúan en *modo texto*; es decir, el usuario sólo puede visualizar caracteres de un juego preestablecido (ASCII ampliado, por ejemplo, véase Apéndice). El controlador de vídeo se encarga de "traducir" el código binario correspondiente a cada carácter en la matriz de puntos de imagen que lo visualiza en la pantalla. En los **monitores gráficos** el usuario tiene acceso al punto de imagen pudiendo representar en ellas dibujos (*modo gráfico*), además de caracteres (*modo texto*).

Los principales parámetros que caracterizan a un monitor de visualización son:

- **Tamaño de la pantalla**. Se da (como en las TV domésticas) en función del tamaño de la diagonal principal, en pulgadas de la parte frontal de la misma, dándose habitualmente en pulgadas. Los tamaños usuales son de 14, 15, 17, 19 y 21 pulgadas.
- **Relación de aspecto**. Es la relación que existe entre el ancho y el alto de la pantalla. Usualmente la relación de aspecto es 4:3; aunque se está introduciendo la relación *panorámica,* de la que hay distintas versiones: 16:9 (británica), europea (10:6), americana (26:14) y CinemaScope o Panavisión (40:17).
- **Número de celdas o caracteres**. Lo usual es una representación de 25 filas x 80 columnas de caracteres (es decir, un total de 1920 celdas), o de 16 filas x 64 columnas (1.024 celdas en total). Algunas pantallas permiten comprimir los caracteres de tal forma que en una línea se pueden visualizar un número mayor de caracteres. Por ejemplo, hay monitores que permiten visualizar 25 x 80 caracteres en modo normal, y 24 x 135 en modo comprimido.
- **Ángulo de visión**. En muchas pantallas (principalmente en las pantallas planas) la calidad de visión depende del ángulo (con respecto a la perpendicular a la superficie de la pantalla) con que se observe la misma. El ángulo de visión es inferior en las pantallas planas que en las pantallas curvas (CRT).
- **Resolución gráfica.** Es el número de píxeles en pantalla en sentido horizontal y vertical. Éste es uno de los parámetros más importantes para determinar la calidad de visualización de imágenes. En la Tabla 11.7 se dan ejemplos de distintos tipos de resolución.
- **Gama de colores.** Es el número de colores o tonos que puede tomar cada punto de la pantalla.
- **Resolución óptica (Densidad y tamaño de puntos de imagen).** La resolución gráfica viene determinada por el número de puntos de imagen en filas y columnas, por lo que no depende del tamaño de la pantalla. Como consecuencia de lo anterior, para una resolución gráfica dada, cuanto más pequeña es la superficie de la pantalla, más próximos aparecerán los puntos dando la imagen mayor sensación de continuidad en sus trazos. Por el contrario, cuanto mayor es la pantalla menor es la calidad de la imagen. Por ello un parámetro de interés es la **resolución óptica** de la pantalla, que es la densidad lineal de puntos de imagen se suele dar en puntos/pulgada (d/i, o dpi, *dots per inch*). Hay que tener en cuenta que el ojo humano es capaz de resolver 380 d/i en una imagen vista a una distancia de 45 cm. Un texto impreso de una revista alcanza una resolución de 1.200 d/i. Uno de los objetivos tecnológicos de las pantallas grandes es conseguir densidades de grabación de 300 d/i.

La distancia entre los centros de dos puntos adyacentes del mismo color se denomina **paso del punto** (*dot-pich*), y es la inversa de la densidad de puntos de pantalla. Cuanto menor sea esta distancia, mejor será la calidad de la imagen aumentando su nitidez, y conviene que sea de 0,26 mm o menor. Los tamaños de los puntos son menores que el paso ya que existe una separación entre ellos (Figura 11.9).

EJEMPLO 11.7

Determinar el tamaño máximo de punto de imagen y de punto de pantalla con una resolución Súper VGA en una pantalla de color de 14 pulgadas.

Se supone una relación de aspecto de la pantalla de 4/3, con lo que, llamando ***a*** *a la anchura de la imagen,* ***h****, a la altura de la imagen y* ***d*** *a la diagonal, se verifica:*

$$a^2 + h^2 = d^2 \quad ; \quad a^2 + h^2 = 14^2 \quad ; \quad \frac{a}{h} = \frac{4}{3}$$

Resolviendo el sistema anterior se tiene que a =11,2"=284 mm y h=8,4"=213mm.

Para obtener una resolución Súper VGA, debe haber 600 puntos de imagen por fila, con lo que cada punto de imagen deberá ser menor que:

$$\frac{284\,mm}{600} = \ = 0,47\,mm$$

Como la pantalla es de color, cada punto de imagen deberá contener horizontalmente, por término medio, al menos 1,5 puntos de pantalla (véase Figura 11.9a), con lo que cada punto de pantalla deberá ser menor que:

$$\frac{0,47\,mm}{1,5} = 0,32\,mm$$

Según se ha comentado con anterioridad, la imagen en una pantalla se forma por medio de puntos de pantalla. Uno o varios puntos de pantalla configuran un punto de imagen o unidad de visualización. Para configurar una imagen se activan selectivamente distintos puntos de pantalla dentro de un cuadro determinado. En unas pantallas la activación de un punto provoca la emisión de luz (autoiluminación) por el punto, y en otras cada punto de imagen actúa como una válvula de la luz ambiental o de la luz de fondo, de forma que la activación del punto controla que se transmita o refleje por él más o menos intensidad de luz.

Por lo general la calidad de las imágenes producidas por las pantallas emisivas es muy superior a la de las pasivas. Según se activen más o menos los puntos de pantalla, así producirán (o reflejarán) mayor intensidad de luz. Es decir, para tener una imagen con distintas tonalidades grises y gamas de colores es necesario que la señales de activación sean analógicas.

La superficie externa de la pantalla (por donde el usuario observa la imagen) puede ser curva o plana. Las pantallas tradicionales contienen un **Tubo de Rayos Catódicos** (**CRT** o *Catode Ray Tube*), similar al de los receptores de televisión, y son de tipo curvo. Debido a su principio físico de funcionamiento, las pantallas planas tienen menos fondo, siendo por tanto mucho menos voluminosas que los CRT, y son mucho menos pesadas, por lo que se utilizan preferentemente para computadores portátiles.

En la Tabla 11.8 se muestra una clasificación de los tipos más importantes de pantallas. Algunos de ellos están en proceso de investigación o desarrollo, y aún no han sido comercializados.

Tabla 11.8. *Distintas tecnologías utilizadas para pantallas de computadores.*

Superficie externa	Fundamento de visualización	Acrónimo	Tipo
Curvas	Tubo de rayos catódicos	CRT (*Catode Ray Tube*)	Emisivas
Planas (FPD)	Cristal líquido (LCD)	TN-LCD (Twisted nematic field effect), ≈20 filas STN-LCD (Supertist nematic, 480 filas) PM-LCD (Passive-Matrix) AM-LCD (Active-Matrix) F-LCD (Ferroelectric) PA-LC (Plasma-addressed liquid crystal)	Pasivas
		TFT-LCD (Thin Film Transistor)	Emisivas
	Plasma	AC-PDP (AC thin-film)	
	Electroluminiscente	AC-TFEL (AC thin-film)	
	Efecto de campo	FED (Field-emissions displays)	

A continuación se describen brevemente los distintos tipos de pantallas.

11.3.2.1 Pantallas de tubo de rayos catódicos (CRT)

La imagen de una pantalla de rayos catódicos se forma al incidir un haz de electrones (*spot*) sobre la superficie interna de la pantalla, que está recubierta de un material fosforescente (Figura 11.11). Dependiendo del tipo de este material se tienen distintas persistencias de imagen y colores. Los colores usuales (en una pantalla monocromática) son el blanco y negro (igual que el de los aparatos de TV en blanco y negro), verde, y ámbar. En las pantallas de color se utilizan tres tipos de fósforos (fósforo rojo, fósforo verde y fósforo azul), que se distribuyen en forma puntual (**luminóforos**) y alternativa a lo largo de las distintas direcciones de la pantalla (Figura 11.11a). Las pantallas CRT hacen que el haz de electrones barra la superficie interna de visualización de la pantalla, de izquierda a derecha y de arriba a abajo y, dependiendo de la intensidad con que inciden los electrones en la pantalla, así de brillante será cada punto de la imagen.

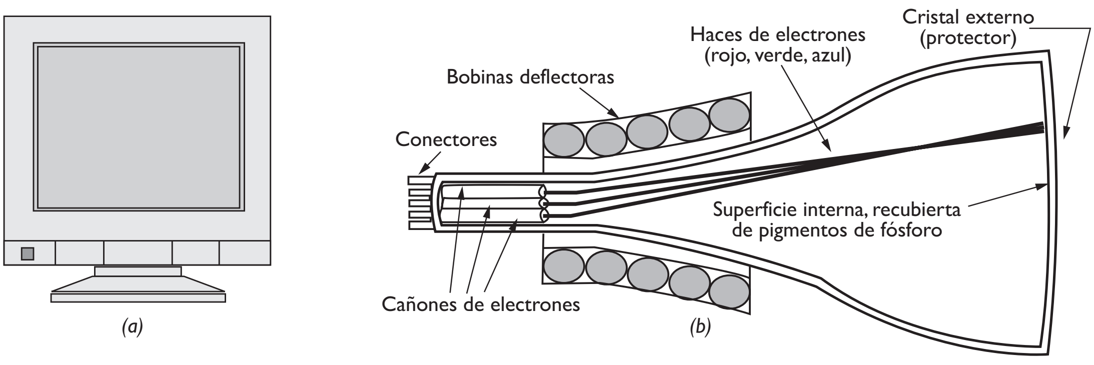


Figura 11.11. *(a) Aspecto de una pantalla CRT; (b) esquema simplificado de una pantalla CRT.*

Para lograr que el haz efectúe el barrido a lo largo de la superficie de visualización, se sitúan externamente al tubo unas bobinas magnéticas (**bobinas deflectoras**) que crean dos campos magnéticos perpendiculares, en direcciones X e Y. Los campos magnéticos desvían el haz de electrones en los sentidos *X* e *Y*. Aplicando unas tensiones adecuadas (en forma de *diente de sierra*) a las bobinas deflectoras se consigue que el haz de electrones recorra la superficie desde izquierda a derecha y de arriba a abajo, barriendo de forma ordenada todos los puntos de pantalla. Simultáneamente a efectuar el barrido, la intensidad del haz de electrones se va modulando con una señal analógica procedente del controlador y que corresponde a la intensidad deseada para cada punto de pantalla.

En las pantallas de color se generan tres haces de electrones simultáneos y cuya intensidad se controla de forma independiente. Cada haz corresponde a un color básico (rojo, verde o azul), y debe incidir exactamente en los puntos de fósforo de su correspondiente color. Las salidas de los amplificadores del controlador de vídeo se aplican directamente en los electrodos que controlan la intensidad de cada cañón electrónico (Sección 11.3.2.3). En los monitores **trinitrón** (desarrollados por Sony) hay un único haz de electrones que, con ayuda de una máscara con ranuras, incide selectivamente en los luminóforos de cada color. En estas pantallas el paso del punto es de 0,25 mm.

11.3.2.2 Pantallas planas

Las pantallas de rayos catódicos (CRT) son las más utilizadas ya que producen imágenes con buen brillo, buen contraste, buena resolución, amplia gama de colores, son robustas, y las imágenes pueden observarse con calidad dentro de un gran ángulo de visión. No obstante, presentan los siguientes inconvenientes: ocupan mucho espacio, son pesadas, consumen mucha energía, y producen rayos-X y campos magnéticos de baja frecuencia, que resultan dañinos para la salud.

La pantallas planas (**FPD**, *Flat-Planel Displays*) presentan las siguientes ventajas: son ligeras, delgadas (poco volumen, Figura 11.12a), y tienen un bajo consumo de potencia. Estas características las hacen preferibles a las CRT para la construcción de computadores portátiles, por ejemplo.

Generalmente las pantallas planas constan de dos cristales planos unidos a presión, entre los que se ubican los elementos activos (Figura 11.12b).

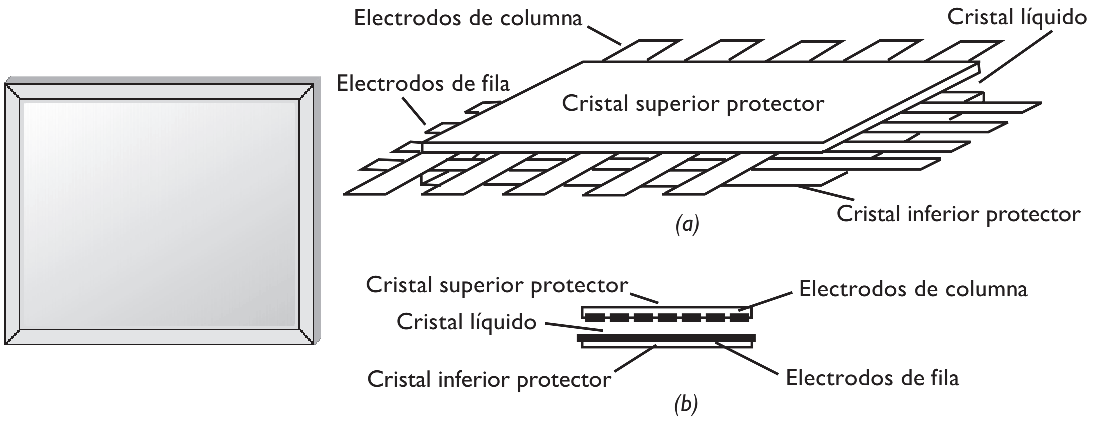


Figura 11.12. *Esquema simplificado de una pantalla de cristal líquido: (a) vista general; (b) corte vertical.*

Las pantallas FPD de alta resolución utilizan un esquema matricial para seleccionar los elementos de imagen utilizando dos juegos de electrodos conductores transparentes en forma de bandas/tiras (Figuras 11.9 y 11.12b). Uno de los juegos se sitúa en sentido horizontal (electrodos horizontales o de filas) y el otro en sentido vertical (electrodos verticales o de columna), y entre ellos se colocan los elementos físicos emisores o reguladores del paso de luz. Los puntos/zonas donde se cruzan los electrodos perpendiculares definen los **puntos de pantalla** (*dot pitch*), direccionables eléctricamente por fila y columna.

EJEMPLO 11.8

Una pantalla VGA monocroma utiliza 640 electrodos o bandas verticales y 460 electrodos horizontales, obteniéndose así 640 x 480 puntos de imagen. Para VGA color el número de electrodos se multiplica, ya que cada punto de imagen está formado por al menos tres subpuntos: rojo, verde y azul (Figura 11.9b), siendo necesarios electrodos independientes para cada uno de ellos; es decir, se necesitarían al menos 921.600 subpuntos con resolución VGA.

Hay dos tipos de matrices de visualización: pasivas y activas.

En las **matrices de visualización pasivas**, las tensiones correspondientes se van aplicando fila a fila (una tensión común en el electrodo de cada fila). Mientras se mantiene una tensión en un electrodo de fila (y_j), se hace un barrido de tensiones por los electrodos de columna (x_i). Los valores de estas tensiones están en consonancia con el brillo deseado en el punto de la fila (x_i, x_j). Cada punto de imagen es, por tanto, activado sólo durante una fracción del tiempo total de formación del cuadro de imagen, parpadeando a la frecuencia de cuadro.

En las pantallas con **matrices de visualización activas**, el parpadeo se reduce y el brillo mejora, considerablemente. Esto es debido a que el punto de imagen se mantiene activado durante la práctica totalidad del periodo de cuadro. Esto se logra gracias a que en cada punto de imagen hay un conmutador electrónico –diodo, transistor MOS, o **transistor de película delgada**, **TFT** (*thin-film transistor*)–, cuya tensión de puerta es refrescada y actualizada a través de los electrodos de columna. En definitiva, la matriz de visualización electrónicamente es como una DRAM, comportándose cada punto de imagen como una capacidad que es recargada o refrescada con el siguiente cuadro. Precisamente en la actualidad el tipo de pantalla más deseado es el **TFT**, debido a su gran contraste y calidad de color; no obstante una de estas pantallas con resolución súper VGA (SVGA) requiere 2.414.592 transistores, lo que presenta una gran complejidad y dificultad de producción a precios competitivos frente a las pantallas CRT.

De acuerdo con las descripciones anteriores, las señales R (rojo), G (verde), V (azul) generadas por el controlador de vídeo (Sección 11.3.2.3) deben aplicarse a los electrodos multiplexadas en el tiempo y en sincronismo con las señales de refresco.

En la Tabla 11.8 se enumeraron los principales tipo de pantallas planas. Las más utilizadas en la actualidad son del tipo de **cristal líquido** o **LCD** (*Liquid Crystal Display*). Utilizan un cristal líquido; es decir, una sustancia oleaginosa que contiene moléculas (cianobifenil) en forma de pequeñas varillas o barras, que se sitúa entre los juegos de electrodos X e Y (Figura 11.12b). En estado normal el cristal líquido es transparente; pero si en una zona de él se aplica un campo eléctrico se vuelve opaco; concretamente las moléculas reaccionan ante los campos eléctricos, reorientándose a lo largo de las líneas del campo; así pueden transmitir

o bloquear punto a punto el paso de la luz para formar la imagen. El problema de estas pantallas es que no se refrescan con una adecuada frecuencia, y si, por ejemplo movemos el ratón, el cursor que le sigue en la pantalla desaparece. La frecuencia tampoco se puede subir arbitrariamente debido al elevado tiempo que tardan las moléculas en establecerse en su posiciones correctas (tiempo de establecimiento). Obviamente el periodo de refresco debe ser mayor que el tiempo de establecimiento. El problema se resuelve con pantallas de doble barrido (*dual scan)* con las que se duplican los circuitos de barrido, y la mejor solución es, como se ha indicado anteriormente, utilizar un transistor de película delgada por cada punto de imagen, ya que de esta forma se mantiene fija la tensión correspondiente a cada punto hasta que se refresque (pantallas TFT).

EJEMPLO 11.9

Una pantalla plana[6] puede tener las siguientes características:

- *Tecnología: TFT.*
- *Tamaño: 30".*
- *Resolución: 1.280 x 1.024.*
- *Brillo: 550 candelas/m^2.*
- *Contraste: 600:1.*
- *Ángulo de visión. Horizontal 170º; vertical: 176º.*
- *Tiempo de respuesta: 12 ms.*
- *Número de colores: 16,8 millones.*
- *Frecuencia horizontal: 31-66 Hz.*
- *Frecuencia vertical: 56-85 Hz.*
- *Tipo de señal de vídeo: RGB analógico/digital.*

11.3.2.3 Controladores de vídeo

La mayor parte de monitores de vídeo no activan los puntos de pantalla de forma continua, sino que lo hacen de forma periódica y durante un corto intervalo de tiempo. Esto se hace, como se al inicio de esta Sección 11.3.2, bien por el principio físico en que se fundamenta la pantalla bien para reducir el consumo de energía eléctrica. La actualización periódica de la información de los puntos de pantalla se denomina **refresco** de la pantalla, e implica un recorrido o **barrido** electrónico de los puntos de pantalla, usualmente de izquierda a derecha y de arriba a abajo (tomando como referencia la imagen que percibe el usuario). La **frecuencia de refresco** suele ser de F_r = *50, 60, 70* ó *90 Hz* (periodo $T_r = 1/F_r$). Un observador no nota el *parpadeo* a esta frecuencia (precisamente el cine se fundamenta en que el ojo humano percibe como continuas las imágenes pulsantes de más de 25 Hz), pero al cabo del tiempo sufre una fatiga que será menor cuanto mayor sea la frecuencia de refresco. La imagen que se visualiza en un periodo T se denomina **cuadro**; esto es, decir que la frecuencia de barrido es 50Hz, equivale a decir que hay que generar un cuadro cada $T = 1/50 = 20$ *ms.*

En la Figura 11.13 se muestra un esquema simplificado de un controlador de vídeo de un monitor de textos. Los registros y elementos internos a los que se accede desde el bus de datos se direccionan como puertos (Sección 9.5.2). Los códigos de los caracteres que van

[6] LCD-TFT LG L3020T.

llegando son analizados por los circuitos que constituyen el **controlador de pantalla**. La mayoría de caracteres que llegan son para visualizar en la pantalla y el controlador de vídeo los almacena directamente en una memoria denominada memoria de vídeo. La **memoria de vídeo** (también denominada **memoria** o **buffer** de **cuadro** o de **exploración**) contiene una reproducción digital de la imagen de pantalla que es leída iterativamente para que la imagen se mantenga (sea "refrescada") en la pantalla. Cada palabra de la memoria de cuadro corresponde, por tanto, biunívocamente a una celda de la pantalla. Por consiguiente, la memoria de cuadro tiene tantas palabras como celdas. Otros caracteres que llegan al monitor son de control (salto de línea, tabulación, etc.), y el controlador de pantalla genera la información o señales de control oportunas. Cada palabra de la memoria en modo texto suele ser de 2 Bytes y almacena, además del código del carácter (Unicode, ASCII, EBCDIC, etc, 8 bits) a visualizar en la celda correspondiente, el **atributo** del carácter: si es intermitente o no (1 bit), el color de fondo (3 bits), y el color de primer plano (4 bits).

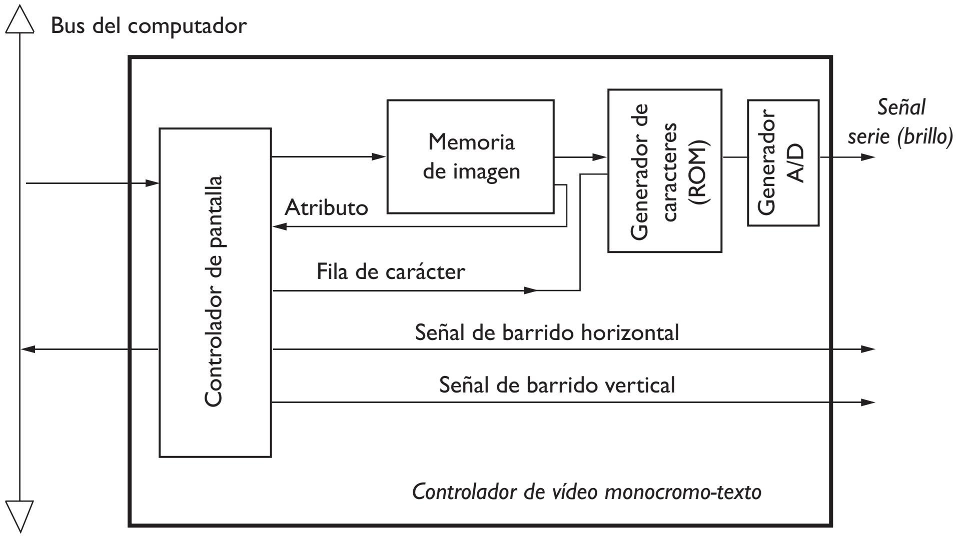


Figura 11.13. *Esquema simplificado de un controlador de vídeo monocromo modo texto.*

EJEMPLO 11.10

Obtener el valor de la capacidad de la memoria de cuadro de una pantalla de 25x80 celdas, con resolución de color EGA, y atributos especiales de parpadeo, vídeo inverso, iluminación tenue y sobreiluminación.

Según se indicó en la Tabla 11.7, los adaptadores de vídeo EGA tienen una resolución de 320x200 y una paleta de 4 colores. Por tanto, el número de palabras de la memoria será:

$320 \cdot 200 = 6{,}25$ *Kpalabras*

Cada palabra debe tener los siguientes bits:

8 bits de código del carácter + 2 bits de color + 4 bits de atributos = 14 bits

Con lo que la memoria será de la siguiente capacidad:

$C = 6{,}25 \cdot 15 = 87{,}5$ *Kbits = 11 KBytes*

La salida de la memoria de imagen se lleva a una memoria ROM denominada **tabla** o **ROM de patrones de caracteres** (Figura 11.13), que contiene una tabla con la representación o forma correspondiente a cada carácter, en términos de puntos de imagen. Cuando el código de un determinado carácter se sitúa en la entrada de la memoria ROM, a la salida de ésta aparece en binario el valor de la intensidad luminosa (señal de brillo) de una de las filas de la matriz de puntos que configura el carácter. En otras palabras, la entrada a la memoria ROM se compone de dos apartados: código de carácter y fila del carácter. Con esta información y la correspondiente a los caracteres de control, el controlador de pantalla compone las tensiones de activación de cada línea de puntos de imagen a visualizar en pantalla (**microlínea**). El controlador de pantalla también, a partir del atributo de cada carácter, genera los colores básicos y señales de barrido de la pantalla de forma similar a como se ve a continuación para los monitores gráficos.

En la Figura 11.14 se muestra un esquema simplificado de un controlador de vídeo de un monitor gráfico. Aquí cada posición de memoria de cuadro, en vez de almacenar la información correspondiente a una celda, almacena la información de cada punto de imagen; es decir, todos los puntos de imagen son direccionables. La información de un punto de imagen, almacenada en código binario en cada palabra de memoria de imagen, corresponde a los niveles de los tres colores básicos y a otros bits para codificar características tales como si el punto de imagen está intermitente o no, si está sobreiluminado o no, etc.

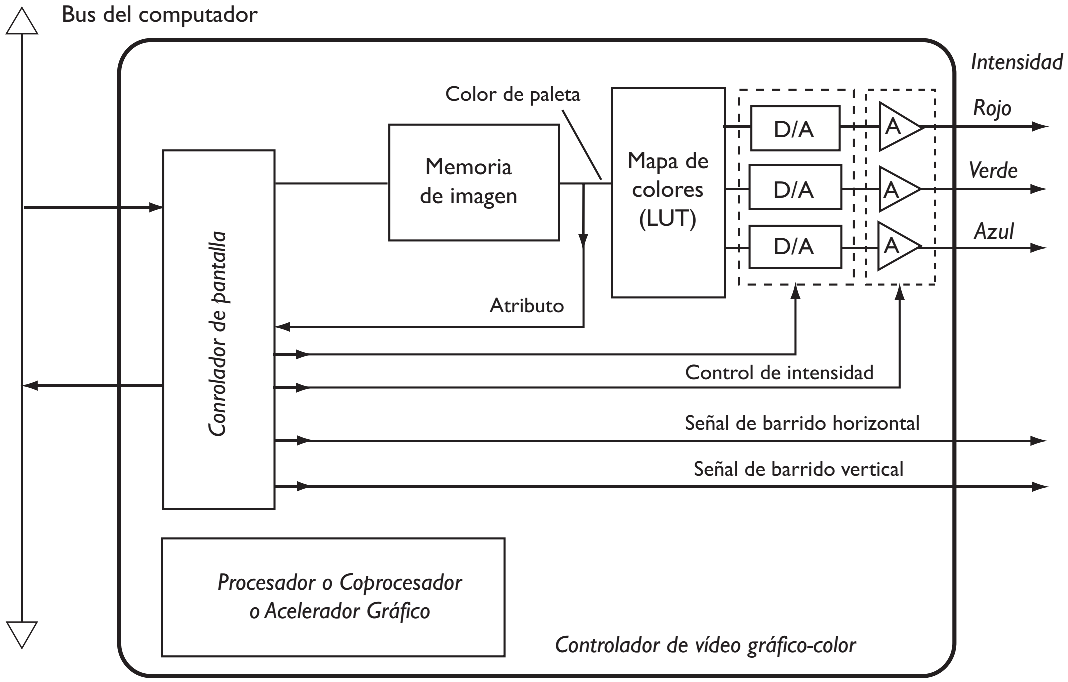


Figura 11.14. *Esquema simplificado de un controlador gráfico de vídeo color.*

La capacidad de la memoria de cuadro crece con el número de puntos de imagen (resolución) y con el número de bits por punto de imagen (paleta de colores o escala de grises, y atributos), como se pone de manifiesto en el ejemplo que se incluye a continuación.

EJEMPLO 11.11

La propuesta de TV de alta definición requiere una resolución de 1.280 x 1.024 puntos de imagen, con 256 niveles por cada uno de los colores básicos. Determinar la capacidad de la memoria de cuadro que se necesitaría para este sistema.

Cada color necesita una codificación de 8 bits ($2^8 = 256$); es decir, cada palabra de la memoria debe contener:

8 bits/color x 3 colores = 24 bits = 3 Bytes

Por otra parte el número de palabras de la memoria debe coincidir con el número de puntos de imagen; es decir:

C = 1.280 x 1.024 palabras = 1.280 Kpalabras de 3 Bytes = 3.840 KBytes = 3,75 MB

Supóngase (como en el Ejemplo 11.11) que para obtener una buena calidad de imagen se requieren poder realizar mezclas con 256 niveles de cada color. En principio se requerirían 24 bits para almacenar el color de cada punto de imagen; con lo que se tendría un total de 2^{24} = 16.777.216 colores posibles. Este número de combinaciones resulta excesivo; pero tal y como indicamos en la Sección 4.7 se puede obtener una calidad casi fotográfica utilizando una **paleta de colores** constituida, por ejemplo, por tan sólo 128 de las 2^{24} mezclas, utilizando, eso sí, las mezclas adecuadas entre las 256 tonalidades posibles de cada color básico. En otras palabras, para muchas aplicaciones es suficiente una paleta de 128 ó 256 colores, pero generándolos con combinaciones elegidas entre 256 niveles de cada color básico. Utilizando este criterio puede conseguirse un ahorro considerable de la memoria de cuadro. En efecto, en la memoria de cuadro se almacenaría el *código de la mezcla* de colores (esto es, el valor dentro de la paleta), y la salida de la memoria de cuadro se lleva sobre una memoria estática, denominada **tabla** o **mapa de colores** o *look-up table* (abreviadamente **LUT**) donde se encuentran almacenadas las mezclas correspondientes a cada valor de la paleta (Figura 11.14). Si se desease una paleta de 256 mezclas y 256 niveles por cada color, la LUT debería tener 8 entradas y 24 salidas (256 palabras de 24 bits). En el caso del Ejemplo 11.11 (TV de alta resolución) la memoria de cuadro se reduciría a una tercera parte.

La salida de la tabla de colores (LUT) es digital, y con ella se siguen obteniendo los colores básicos de cada punto de imagen en forma codificada. Estas señales, por medio de unos conversores digital/analógico (D/A, Sección 11.5.2), y amplificadores se transforman para actuar sobre las entradas de color del monitor de vídeo.

El controlador de pantalla o vídeo también se encarga de generar las señales de exploración o refresco de la imagen. El controlador de pantalla para un CRT se suele denominar CRTC (*Catode Ray Tube Controler*).

En la memoria de cuadro se realizan dos operaciones:

- *Actualización con imágenes nuevas,* lo que se realizará dentro de un determinado tiempo T_a. El valor máximo de T_a depende de la aplicación. Si la aplicación es de edición de textos, para no dar una sensación desagradable de lentitud al usuario, se requiere una frecuencia de cambio de la memoria de cuadro del orden de F_a=5 cuadros/segundo (T_a = 0,2s). Si la aplicación implica visualizar figuras con movimientos suaves y de cierta calidad (calidad TV, para sistemas multimedia, por ejemplo) es necesario cambiar la *imagen contenida en la memoria de cuadro a una frecuencia*

del orden de F_a=24 cuadros/segundo (T_a= *0,042 s*). Si la aplicación es de tipo CAD, es conveniente que la imagen (o parte de ella) cambie de forma casi instantánea; por lo que el tiempo de actualización debe ser menor, ya que el ojo humano considera instantáneos los cambios realizados en menos de T_a=*0,02 segundos*.

- *Refresco de la pantalla.* Según se indicó en el inicio de la Sección 8.2, la frecuencia de refresco, F_r, suele estar comprendida entre *50 y 100 Hz*. Esto implica que cada T_r segundos (*0,02 s*<T_r< *0,012 s*) debe refrescarse la pantalla.

Para realizar la actualización de la memoria de cuadro y su lectura para refresco de pantalla, a lo largo del tiempo se han ido utilizando tres técnicas consistentes en efectuar la *actualización de la pantalla entre dos refrescos* sucesivos, utilizar una *doble memoria de cuadro* (mientras se está actualizando una de ellas, la lectura o refresco se hace con la otra), y utilizar *memorias bipuerto*. Existen memorias RAM dinámicas *bipuerto* ideadas para los controladores de vídeo (**DRAM de vídeo**) que tienen, además de la E/S para la información a leer/escribir aleatoriamente, otra E/S para poder leer/escribir en la memoria de forma secuencial. Esta última E/S se conecta a las etapas que van hacia la pantalla, ya que la pantalla siempre se refresca de forma secuencial. La otra E/S se conecta hacia la entrada del controlador, ya que la actualización debe poder realizarse con acceso aleatorio.

La técnica utilizada, la frecuencia de actualización de imágenes, y la frecuencia de refresco de la pantalla, junto con la capacidad de la memoria de cuadro determinan las velocidades de transferencia o **ancho de banda** de datos de dicha memoria, como se pone de manifiesto en el siguiente ejemplo.

EJEMPLO 11.12

El ancho de banda entre una memoria y los circuitos de control de la pantalla necesario para refrescar la pantalla 30 veces por segundo con resolución XVGA será:

Capacidad de la memoria:

$$C_M = 1.024 \cdot 768 \cdot 4 \text{ Bytes/píxel} = 3 \text{ MB}$$

Donde hemos tenido en cuenta que la resolución XVGA es de 1.024x768 píxeles, y que por cada píxel se almacenan 32 bits (8 bits por cada color básico y otro byte de reserva).

Como hay que transmitir la imagen 30 veces por segundo, el ancho de banda será:

$$AB = C_M \cdot 30 \text{ Hz} = 3 \text{ MB} \cdot 30 \text{ Hz} = 90 \text{ MB/s}$$

Un monitor es un periférico de salida, pero en realidad es considerado por el procesador como un medio de entrada y salida ya que el procesador (o DMA) es capaz tanto de leer como de escribir en la memoria de cuadro; esto es debido a que algunas operaciones con imágenes de pantalla (por ejemplo, desplazar una ventana) son hechas directamente por el procesador, actuando éste sobre la memoria de vídeo. La lectura de la memoria de vídeo se efectúa utilizando la dirección del puerto correspondiente.

Los controladores gráficos actuales contienen un **procesador** o **coprocesador** o **acelerador gráfico** (Figura 11.14). Estos circuitos pueden constituir un auténtico microcomputador de uso específico, con un microprocesador especializado en realizar primitivas más o menos complejas de dibujo. Así, pueden realizar funciones tales como zoom de la imagen, dibujo de ventanas, rellenado de formas (dar textura), desplazamiento rápido de

bloques de información (para movimiento de iconos o ventanas), etc. Hay procesadores gráficos de tres dimensiones (3D), en los que en cada posición de memoria de cuadro se almacena la *profundidad* (un valor **z**) del punto de imagen. El color del punto se reajusta en función del valor del campo z, para dar al usuario la sensación de profundidad. Los términos de coprocesador o procesador se utilizan según que el circuito correspondiente realice tan sólo funciones delegadas por el procesador, o funciones más complejas programables directamente por el usuario, respectivamente.

EJEMPLO 11.13

Un procesador gráfico[7] comercial puede tener las siguientes características:

- *Arquitectura superescalar de 16 canales de procesamiento (Sección 12.7).*
- *Bus: PCI Express (> 4GB/s) (Sección 12.5).*
- *Color: 32 bits.*
- *Memoria: 2 módulos RAMDAC a 400 MHz. Una memoria RAMDAC (Random Access Memory Digital-to-Analog Converter) es una memoria SRAM cuyas entradas son binarias y tiene en la salida tres conversores D/A que proporcionan las señales analógicas RGB para actuar directamente con la pantalla (moduladores de los haces de electrones en una CRT).*
- *Refresco: hasta 85 Hz para 2.048x1.536 píxeles.*
- *Permite conectar pantallas planas grandes con una resolución máxima de 1.900x1.200 píxeles.*
- *Codificador integrado para TV de alta definición, con resolución máxima de: 1.920x1.200 píxeles.*

11.3.3 Impresoras

Las impresoras son periféricos que escriben la información de salida (caracteres o puntos que forman una imagen) sobre papel. Es decir, permiten visualizar en forma impresa (permanente) los datos o resultados generados por el computador. Su comportamiento inicialmente era muy similar a las máquinas de escribir, pero hoy día son mucho más sofisticadas, pareciéndose algunas en su funcionamiento a máquinas fotocopiadoras conectadas en línea con el computador.

Las impresoras son, junto a las pantallas, los dispositivos más utilizados para poder ver en forma directamente inteligible para el hombre los resultados de un programa de computador.

Como se indicó en la Sección 11.1 para todos los periféricos, las impresoras tienen dos partes diferenciadas: parte mecánica y electrónica. Aquí la parte mecánica, además de encargarse de accionar los elementos que hacen que se imprima el carácter correspondiente, debe dedicarse a la alimentación y arrastre del papel. Las primeras impresoras utilizaban papel continuo con márgenes taladradados; en este caso el arrastre se efectuaba por un tractor que disponía de unos dientes metálicos o de plástico que encajaban en los taladros laterales del papel. En la actualidad la mayoría de impresoras no necesitan papel continuo, efectuándose el arrastre por fricción o presión, como en el caso de las máquinas de escribir o como en las fotocopiadoras convencionales.

Una cuestión común a las impresoras de color es cómo producen éste. El procedimiento es similar al de las pantallas, es decir, un color concreto se genera por mezcla de colores; ahora

[7] Nvidia Geforce 6200.

bien, en el caso del papel el color que se percibe por el ojo humano no es el que generado por una fuente de luz (filtrado o no) sino el que *refleja* el papel, esto es, el no absorbido por la tinta impresa; por lo que habrá que utilizar colores sustractivos. Otra cuestión a tener en cuenta es que, en general, no se dispone de colorantes que produzcan tres colores básicos de calidad, por lo que se suelen utilizar cuatro: uno para el negro (denominado color K de *blacK*) y los otros tres: azul (o C, de Cyan), amarillo (o Y, de Yelow) y rojo (o M, de Magenta). Este conjunto de colorantes se suele denominar **CYMK**.

Existen diversos criterios para clasificar las impresoras, según se especifica en la Tabla 11.9.

Tabla 11.9. *Clasificación de impresoras según diversas características.*

Criterio	Clasificación
Calidad de impresión	– Normal (matriciales, térmicas)
	– Semicalidad (matriciales, inyección de tinta)
	– Calidad (inyección de tinta, sublimación, láser)
Fundamento del sistema de impresión	– Impacto (matriciales)
	– No impacto (inyección de tinta, láser)
Forma de imprimir los caracteres	– Impresoras de caracteres (matriciales, inyección de tinta)
	– Impresoras de líneas (térmicas)
	– Impresoras de páginas (láser e inyección de tinta)

Fundamento del sistema de impresión

Hay unidades que realizan la impresión por impacto de martillos o de piezas móviles y otras sin impactos mecánicos. Éstas se denominan **impresoras de impacto**, y son muy ruidosas, pero pueden hacer copias múltiples utilizando papel especial, y tradicionalmente han sido las más utilizadas. Entre ellas se encuentran las impresoras de rueda, bola, margarita, cilindro, cadena, fleje de acero y matriciales. Las **impresoras sin impacto** forman los caracteres sin necesidad de golpes mecánicos y utilizan otros principios físicos para transferir las imágenes al papel. En este grupo se encuentran las impresoras térmicas, de inyección de tinta, de transferencias electrostática y magnética, y las impresoras láser, que describiremos brevemente más adelante, en esta misma sección.

Forma de imprimir los caracteres

En cuanto a este aspecto, las impresoras se pueden clasificar en impresoras de caracteres, de líneas, y de páginas.

- **Impresoras de caracteres.** Realizan la impresión por medio de un cabezal que va escribiendo cada línea carácter a carácter (en un instante dado sólo se escribe un carácter). El cabezal se desplaza a lo largo de la línea que se está imprimiendo, de izquierda a derecha, (impresoras unidireccionales), o, para conseguir mayor velocidad, en ambos sentidos, sucesivamente, (impresoras bidireccionales). Las velocidades de estas impresoras oscilan entre 10 y 400 caracteres por segundo. Dependiendo del cabezal existen, a su vez, distintos tipos de impresoras de caracteres [Pri95]: impresoras de rueda, de bola, de margarita, de matriz de puntos, térmicas, y de inyección de tinta.
- **Impresoras de líneas.** En estas impresoras, a diferencia de las de caracteres, se imprimen simultáneamente todos o varios caracteres correspondientes a una línea de impresión. Hay que recordar que el procesador (o memoria principal) rellena

rápidamente ("a ráfagas") la memoria intermedia de caracteres, no siendo necesario, por tanto, escribir los caracteres en el orden que llegan uno detrás de otro, de izquierda a derecha; sino, que una vez llena la memoria intermedia (que contiene los caracteres de al menos una línea), la operación mecánica de escribir los caracteres puede realizarse en cualquier orden (todos los caracteres a la vez, o primero todas las A de la línea, luego todas las B, y así sucesivamente, por ejemplo).

La velocidad de impresión de estos dispositivos es muy superior a las de caracteres, oscilando usualmente entre 300 y 2000 líneas por minuto. Suponiendo que, por término medio, se escriban 80 caracteres por línea, estos márgenes equivalen aproximadamente de 400 a 2700 caracteres por segundo. Algunas de las impresoras de líneas más importantes son impresoras térmicas, de tambor, de barra, de banda o fleje, y de transferencia electrostática [16].

- **Impresoras de páginas.** Aquí incluimos un grupo de impresoras que procesan e imprimen considerando cada página como un todo; así, por ejemplo pueden imprimir empezando por los primeros puntos del conjunto de las líneas de la página completa y así sucesivamente. En este grupo se incluyen las impresoras láser y algunas de inyección.

A continuación describiremos los tipos de impresoras más importantes.

Impresoras matriciales o de agujas

También denominadas de matriz de puntos, se utilizan siempre que se requiere imprimir con copias (ya que son de impacto); de este tipo suelen ser las pequeñas impresoras (cajas registradoras, comprobantes de pago con tarjetas de crédito, cajeros de entidades bancarias, etc.). Los caracteres se forman por medio de una matriz de puntos (de 7 x 5, 7 x 9, 8 x 9, 9 x 9, 9 x 11, 18 x 8, ó 24 x 24 etc. –indicamos primero los puntos de altura y luego los de ancho, del carácter), que son creados por agujas o alambres de impresión disparados por electroimanes (Figura 11.15). Hay cabezales de impresión que contienen las agujas en dos dimensiones (Figura 11.15b), pero lo más usual es que estén dispuestas según una línea vertical que contiene 7, 8, 9, 11 ó 24 agujas dispuestas en línea, una encima de otra (Figura 11.15a). Cuanto mayor es el número de agujas mayor será la calidad de impresión (mayor apariencia de continuidad tendrá el carácter impreso). Las agujas, por acción de los electroimanes respectivos, golpean la cinta entintada, trasfiriéndose al papel los puntos correspondientes a las agujas disparadas. Los caracteres son, por tanto, al igual que los de una pantalla, punteados. Su bajo precio hace que estas impresoras sean muy populares. Además existen algunos modelos en los que desde el programa puede seleccionarse arbitrariamente cualquier aguja individual, efectuándose así gráficas y dibujos en forma parecida a una pantalla gráfica.

Una ventaja que presentan algunos modelos de impresoras matriciales es poder seleccionar la calidad de impresión: alta, media y de "borrador", obteniéndose mayores velocidades de impresión cuando la calidad es peor. Con frecuencia la calidad mejor se obtiene sobrescribiendo los caracteres varias veces, con las agujas ligeramente desplazadas, de esta forma se solapan los puntos impresos unos con otros.

Al igual que en el monitor de visualización descrito en la Sección 11.3.2 (Figura 11.10), las formas de los caracteres a escribir, en función de las agujas del cañón de impresión a disparar, están grabadas en una memoria ROM, pudiéndose cambiar la forma (o "**fuente**") de los caracteres sustituyendo los circuitos integrados que constituyen la ROM.

Las velocidades de este tipo de impresoras de caracteres suele oscilar entre 180 y 500 caracteres por segundo.

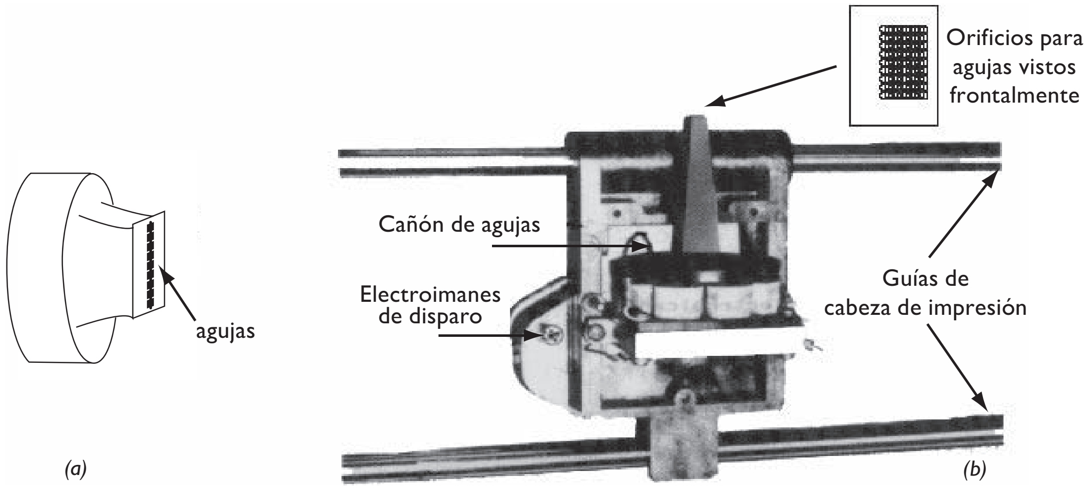


Figura 11.15. *Cabezas de impresoras matriciales: (a) con disposición lineal de las agujas (7 agujas); (b) con la disposición de sus agujas en dos dimensiones.*

Impresoras de banda (o fleje) de acero

Las impresoras típicas de los servicios de informática que requerían gran producción de listados eran de tipo impacto y de línea. Los caracteres se encontraban modelados en ruedas, en tambores, en barras o en cadenas (impresoras de rueda, tambor, barra o cadena, respectivamente [Pri95]), y la forma del molde pasaba al papel al percutir sobre él un martillo. A pesar de su antigüedad se siguen utilizando aún **impresoras de banda de acero**. En éstas, los caracteres se encuentran modelados en negativo sobre la superficie externa de una banda o fleje de acero. Para conseguir mayor velocidad el juego de caracteres se encuentra repetido varias veces (véase Ejercicio 8.4). El fleje se encuentra cerrado y girando constantemente a gran velocidad frente a la cinta entintada. Cuando los moldes de los caracteres a escribir se posicionan delante de las posiciones en que han de quedar en el papel, se disparan por detrás de éste unos martillos, imprimiéndose de esta forma la línea. El número de martillos coincide con el número de caracteres por línea. Las velocidades de impresión suelen estar entre 600 y 2.400 líneas por minuto.

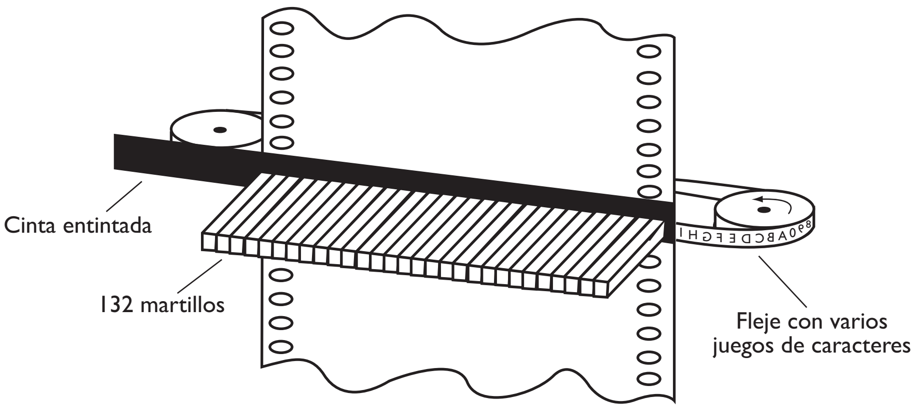


Figura 11.16. *Esquema de una impresora de banda de acero.*

Impresoras térmicas

Las impresoras térmicas se caracterizan porque cada punto del cabezal o sistema de impresión tiene asociado una matriz de pequeñas protuberancias en el interior de cada una de las cuales hay una resistencia eléctrica de caldeo. La temperatura de cada una de las protuberancias determinada la intensidad de la impresión del punto donde esté posicionada. Podemos incluir en este grupo los siguientes tipos:

- **Impresoras de papel térmico.** Son similares a las impresoras de agujas. Se utiliza un papel especial termosensible que se ennegrece al aplicar calor (unos 200°). El calor se transfiere desde el cabezal por la matriz de resistencias. Al pasar una corriente eléctrica por las resistencias se calientan, formándose los puntos en el papel. No requieren ningún consumible, como cinta entintada, tinteros o tóner, salvo el papel. Su velocidad oscila entre unos 100 y 2000 caracteres por segundo. Las lentas son impresoras de caracteres (las líneas se imprimen con un cabezal móvil). Las rápidas son de líneas, y contienen tantas cabezas como caracteres a imprimir por línea, operando todas ellas en paralelo. En la actualidad las más pequeñas son muy utilizadas para emitir recibos.
- **Impresoras de cera**. En este caso los puntos de escritura se forman con cera. Al igual que las primitivas cintas entintadas de color para máquinas de escribir, se dispone de una cinta o banda de gran anchura, formada por tres sub-bandas recubiertas con cera de los colores CYMK. El sistema hace que sucesivamente pasen por encima de la superficie a imprimir las cuatro sub-bandas, y, en cada una de esas situaciones, se aplica un nivel de corriente eléctrica a cada resistencia del punto de imagen de un valor en consonancia con la intensidad con la que deba aplicarse el color correspondiente.
- **Impresoras de transferencia térmica con sublimación.** El sistema es análogo a las impresoras de cera, pero más sencillo ya que en lugar de tener que disponer de la banda encerada, el medio de impresión está constituido por paneles o tabletas CYMK sólidos que van posicionándose en la zona de impresión. En este caso los diminutos y numerosos calefactores hacen que se vaporicen (se *sublimen*) los colorantes sólidos y se fijen, por contacto, en el papel. Como en las impresoras de papel térmico, es necesario un papel especial pero, por el contrario, se obtienen imágenes de color de una extraordinaria calidad (fotográfica) ya que se controla de forma eficiente la cantidad de colorante trasferida en cada punto de imagen (250 niveles) y la mezcla de los colorantes es muy eficiente por realizarse en forma gaseosa.

Impresoras de inyección de tinta

El fundamento físico es similar al de las pantallas de vídeo. En lugar de transmitir un haz de electrones se emite (Figura 11.17) un chorro de gotas de tinta ionizadas, que en su recorrido es desviado por unos electrodos que se encuentran a un potencial fijo (del orden de 3 KV). El carácter se forma con la tinta que incide en el papel. La desviación de las gotas, y por tanto la forma del carácter, se regula variando la carga inicial de la gota dada en un electrodo de carga (el potencial de éste varía de 0 a unos 200 V). Cuando no se debe escribir, las gotas de tinta se desvían hacia un depósito de retorno, como muestra la figura. Para lograr una buena nitidez de la imagen y de los colores que la componen es necesario que el tamaño del punto provocado por cada gota sea lo menor posible, consiguiéndose en algunas impresoras gotas de tan sólo 1 picolitro de volumen.

Este tipo de impresora dispone de uno, caso de impresoras blanco y negro, o un conjunto CYMK de tinteros (impresoras de color) acoplados a uno o varios pulverizadores, a través

de los cuales se dispara (inyecta) la tinta hacia el papel en forma de diminutas gotitas. Las gotas se suelen obtener por uno de los siguientes dos procedimientos:

- **Burbuja térmica** (*Bubble-Jet*). Se calienta instantáneamente la tinta, se vaporiza en forma de burbuja, hasta que explota, arrastrando la tinta del gotero hacia el exterior.
- **Pulverización piezoeléctrica**. El pulverizador contiene un cristal piezoeléctrico que se dilata y contrae al aplicar un campo eléctrico. Al contraerse el gotero se expulsa una gota, y al dilatarse absorbe el fluido del tintero.

Las impresoras de inyección de tinta actuales disponen de cabezas de impresión con hasta 64 aberturas de inyección y se pueden conseguir densidades de impresión tan altas como 2.400 x 1.200 dpi (puntos/pulgada), obteniéndose resultados razonables para gráficos y fotografías de color. Puede decirse que son las impresoras más adecuadas para PC de uso doméstico, ya que son relativamente baratas, silenciosas y muy versátiles.

Existen también **impresoras de tinta sólida,** en las que en lugar de permanecer la tinta en forma líquida, lo hace en piezas sólidas que, al conectar la impresora, se funden poco a poco pasando a los pulverizadores respectivos. Para fijar la tinta al papel, al igual que ocurre con las impresoras láser, es necesario presionar y enfriar el papel inyectado por medio de unos rodillos.

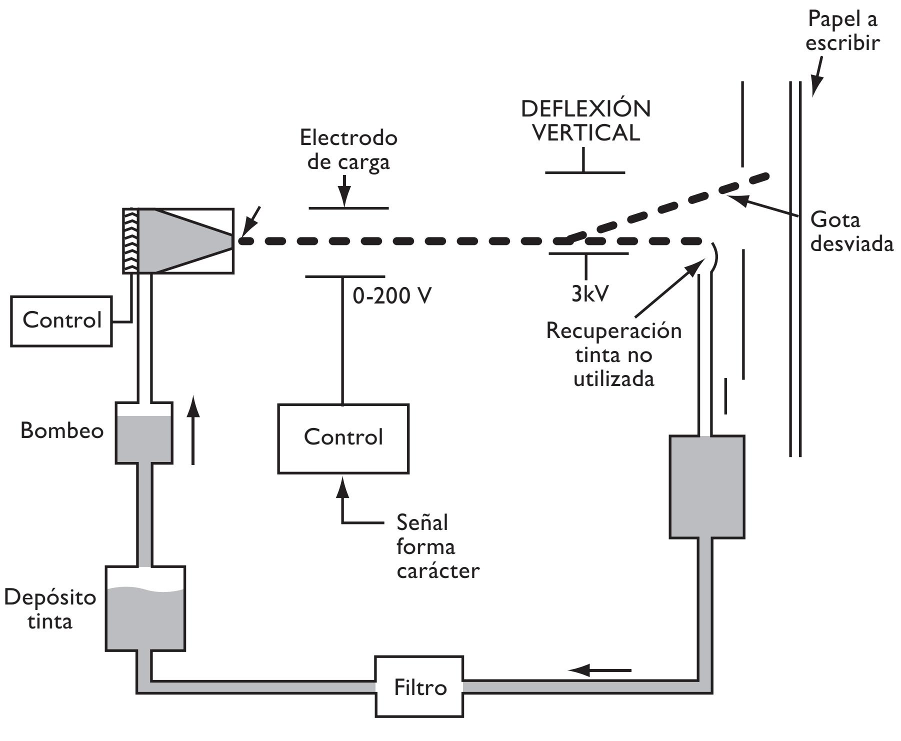


Figura 11.17. *Esquema de una impresora de inyección de tinta.*

Impresoras láser

Las impresoras láser tienen una gran importancia por su elevada velocidad, calidad de impresión, bajo precio, y utilizar, la mayoría de ellas, papel normal (no necesitan ni papel continuo, papel térmico, etc., Figura 11.18a).

La página a imprimir se transfiere al papel por contacto, desde un tambor que contiene una imagen impregnada en **tóner** (polvo de carbón).

El tambor está recubierto de un material fotoconductor (usualmente Selenio), que a oscuras mantiene la carga eléctrica, y con iluminación se descarga. La imagen *eléctrica* se forma en el tambor haciendo incidir sobre él un rayo láser que va barriendo las generatrices del tambor. Cada generatriz suele corresponder a una "columna" de puntos de la página a imprimir; es decir, no se escribe *renglón a renglón* (en la dirección "x"), sino a lo largo del papel (dirección "y"). Una impresora láser tiene tres partes fundamentales (Figura 11.18b):

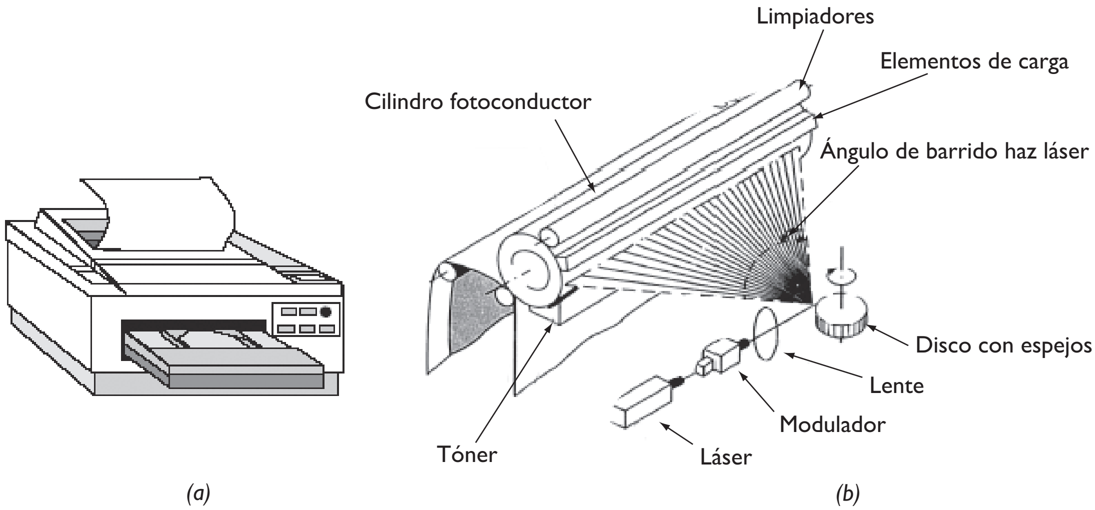


Figura 11.18. *(a) Aspecto externo de una impresora láser de sobremesa; (b) elementos básicos de una impresora láser.*

a) *Elementos mecánicos.* Integrados por el tambor fotoconductor, el sistema de alimentación y arrastre del papel, y tres elementos que actúan directamente sobre el tambor: elemento de limpieza de la superficie (con cuchilla de limpieza), electrodo o corona de carga, que va cargando uniformemente la superficie a un potencial negativo, y elemento de suministro y pulverización de tóner. El haz láser barre una generatriz del tambor en la zona entre el electrodo de carga y la estación de tóner. Los puntos sobre los que incide el haz de láser se descargan, correspondiendo a las zonas o puntos que aparecerán no impresas (en blanco) en la página. El tóner se carga positivamente, de forma que impregnará las zonas de la superficie del tambor cargadas negativamente. A continuación, electrostáticamente, por presión y por calor se transfiere el tóner del tambor al papel, quedando éste escrito.

b) *Elementos ópticos*. En esencia integrados por:
   - Un láser, de Helio-Neón.
   - Un modulador opto-acústico. Con una señal eléctrica puede desviarse (eliminarse) el haz láser de su trayectoria. El rayo láser se elimina en los puntos donde hay que imprimir, ya que así estos permanecen cargados y atraerán el tóner.
   - Una lente que enfoca el haz sobre un espejo.
   - Un disco giratorio con 6 o más espejos. Al ir girando el disco, y mientras incide el haz láser en un espejo, el rayo es desviado de forma que barre todos los puntos

de una generatriz del tambor. Mientras esto ocurre, y para cada punto, la señal eléctrica que actúa sobre el modulador debe o no desviar el rayo de su trayectoria hacia el espejo.

c) Elementos electrónicos. Suministra la señal eléctrica al modulador. Esta señal es digital correspondiendo, por ejemplo, el 1 a eliminación del haz, y el 0 a no eliminación del haz. La imagen se forma por puntos, similarmente a la imagen de una pantalla (Sección 8.3.1). Entre estos elementos se encuentra la memoria intermedia (“buffer”) que suele tener capacidad para almacenar varias páginas.

Los circuitos electrónicos, además de controlar la transmisión de información con el procesador o memoria principal, tienen por objeto crear un “mapa de bits (o puntos) de imagen”. Este mapa contiene información sobre los puntos de la página: para cada uno de ellos, si debe estar escrito (no hay que descargar el punto correspondiente del tambor para que en él se fije tóner) o si debe quedar en blanco (hay que descargar el punto). Esta información, adecuadamente amplificada, es la que actúa como entrada del modulador opto-acústico.

Las impresoras láser de color utilizan cuatro depósitos de tóner correspondientes al conjunto CYMK, produciendo la variedad de colores por mezcla.

Las velocidad de las impresoras láser suele darse en **p**áginas **p**or **m**inuto (en la actualidad los límites se encuentran aproximadamente entre 4 y 350 ppm), obteniéndose resoluciones de impresión tal altas como 1200 dpi.

EJEMPLO 11.14

Una impresora[8] de inyección de tinta puede tener las siguientes características:

- *Tecnología de impresión: inyección térmica de tinta.*
- *Conexión al computador: inalámbrica (802,11b/g), USB, Ethernet.*
- *Resolución máxima: b/n: 1.200 x 1.200 dpi; color: 4.800 x 1.200 dpi.*
- *Velocidad de impresión, calidad borrador: b/n: 30 ppm; 20 ppm.*
- *Velocidad de impresión, calidad óptima: b/n: 2,1 ppm; color: 2,1 ppm.*
- *Memoria temporal. 32MB.*

11.3.4 Registradores gráficos

Los **registradores gráficos** (en inglés: *plotters*) son dispositivos de salida que realizan dibujos sobre papel. Estos periféricos tienen gran importancia ya que con ellos se obtienen directamente del computador salidas en forma de planos, mapas, dibujos, gráficos, esquemas, e imágenes en general.

El funcionamiento de un registrador gráfico se controla desde un programa. El usuario puede incluir en su programa en lenguaje de alto nivel de instrucciones o llamadas a subrutinas para realizar las representaciones que desee con sus datos. El software del computador debe contener rutinas que faciliten la utilización del registrador, y que pueda llamar el usuario dándoles los parámetros adecuados. Las rutinas controlan funciones tales como, subida o bajada de una determinada plumilla, el dibujo y rotulación de ejes, rotulación de figuras, sombreado o entramado de ciertas zonas, forma de escribir las líneas (continuas, punteadas o a trazos), escala, etc.

[8] HP Deskjet serie 6840 (C9031B).

Los registradores gráficos (Figura 11.19) se fundamentan en el desplazamiento relativo de un cabezal con el elemento de escritura, con respecto al papel. Dependiendo del tipo de registro gráfico se moverá sólo la cabeza, o la cabeza y el papel; estos movimientos se controlan por medio de *motores de pasos* de gran precisión que se activan digitalmente.

Hoy en día la importancia de los registradores gráficos ha decrecido ya que en muchos trabajos pueden ser sustituidos por impresoras gráficas, así con una impresora láser se pueden obtener dibujos de una gran calidad. Los registradores gráficos se suelen reservar para dibujos de gran tamaño (A0, por ejemplo) o que requieran una gran diversidad de colores.

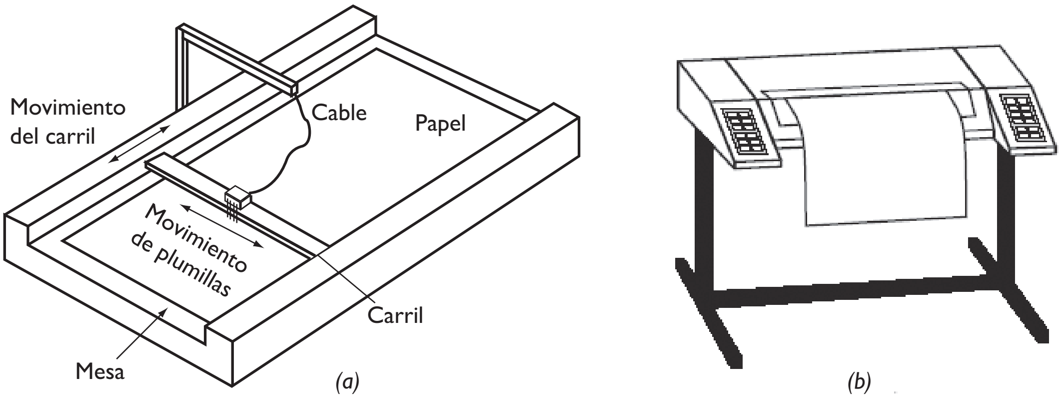


Figura 11.19. *Registrador gráfico: (a) de sobremesa; (b) de tambor.*

11.4 Periféricos para aplicaciones multimedia

Las aplicaciones multimedia son programas en los que se utilizan conjuntamente sonidos, imágenes de vídeo e imágenes producidas por computador.

Una aplicación multimedia, por ejemplo es la consulta por computador de una enciclopedia moderna. La enciclopedia suele estar contenida en CD-ROM. Si consultamos, por ejemplo, la palabra *Mozart* aparece en la pantalla un texto describiendo su vida, probablemente habrá una foto (imagen) de la casa en que nació. Además, al citar en el texto sus composiciones más conocidas, el usuario podrá seleccionar con el ratón una de ellas y escucharla. Es más, sería posible seleccionar un pasaje de una de sus óperas y visualizar una imagen de vídeo de alguna de sus escenas. En definitiva el CD-ROM contiene imágenes de vídeo, sonidos (música), además de los gráficos y sistemas de visualización clásicos de computador, que la aplicación multimedia permite reproducir y combinar adecuadamente.

Las aplicaciones multimedia tienen las siguientes peculiaridades:

- Suelen requerir archivos de gran capacidad, por lo que utilizan las técnicas de compresión de datos vistas en la Sección 4.7.
- Necesitan gran potencia de cálculo, de forma que los repertorios de instrucciones de los modernos microprocesadores suelen incluir conjuntos de instrucciones específicas para mejorar las prestaciones en aplicaciones de imágenes, audio, vídeo 3D y reconocimiento del habla. Este es el caso, por ejemplo, del conjunto SSE del Pentium III [Ort00].
- El bus (o buses) de interconexión entre memoria y controlador de pantalla debe de ser de gran velocidad, sobre todo si se quieren visualizar imágenes en movimiento.

- Debe disponerse de periféricos adecuados. Puede considerarse que los periféricos específicos de multimedia son el sistema de audio, y sistema de vídeo incluyendo una pantalla de alta resolución.

La pantallas se analizaron en la Sección 11.3.2, en las secciones siguientes se describirán brevemente los sistemas de audio y de vídeo.

11.4.1 Sistemas de audio

Las señales de audio, u ondas sonoras, se transmiten en forma de cambios rápidos de la presión del aire. El oído humano es capaz de detectar ondas de sonido de frecuencias comprendidas entre 20 y 20.000 Hz. Los sensores de señales de audio se denominan **micrófonos** y los actuadores **altavoces**. Los primeros transforman las ondas sonoras en señales analógicas, que posteriormente deben digitalizarse con un conversor A/D para ser procesadas por un computador. Los altavoces transforman señales analógicas en señales de audio (generan ondas de presión).

Los aspectos de audio relacionados con los computadores son:

- Síntesis de sonido.
- Reproducción de sonido.
- Síntesis del habla.
- Reconocimiento del habla.

Para captar, procesar y producir sonidos con cierta calidad es necesario disponer de un sistema de sonido que incluya los siguientes elementos:

- **Tarjeta de sonido**, que contiene conversores A/D (entrada) y D/A (para salida), sintetizador de sonidos e interfase MIDI.
- **Altavoces externos**, con bafles, amplificadores y alimentación autónoma.
- **Software específico**, incluye programas de control.

Las tarjetas de sonido están proyectadas para procesamiento de voz o música (sistemas multimedia). Permiten:

- Convertir sonidos almacenados (digitalmente) en señales analógicas audibles por los altavoces.
- Almacenar sonidos para ser procesados y reproducidos posteriormente.
- Crear (sintetizar) sonidos.
- Mezclar sonidos.
- Conectar instrumentos musicales.

Con un sistema de audio se pueden reproducir en forma audible archivos de audio previamente almacenados y, además, sintetizar (generar) sonidos nuevos.

11.4.2 Sintetizadores de sonidos

La mayoría de sintetizadores de sonidos se basan en la idea de que cualquier sonido se puede formar con la superposición de señales sinusoidales o, en general, señales periódicas. Debido a ello para generar sonidos es necesario disponer de un oscilador básico. Una vez realizada la mezcla de señales generadas, ésta debe ser llevada a la entrada de un conversor D/A (que transforma la información binaria en analógica) y, después de amplificada, a un altavoz para transformar las señales eléctricas en sonidos.

La calidad del sonido depende, entre otros factores, del altavoz, principalmente del tamaño y de cómo esté instalado (bafle). En las configuraciones básicas de los PC se dispone de un pequeño altavoz de tan sólo unas 2,5 pulgadas y sin bafle, que produce un sonido de muy baja calidad; la generación de sonidos se realiza a partir del oscilador del sistema que se encuentra en uno de los chipset.

La calidad del sonido mejora sustancialmente utilizando una tarjeta de sonido. Así la tarjeta *Sound Bluster* dispone del circuito YM3812 que puede generas 9 sonidos instrumentales a la vez.

El estándar **MIDI** (*Musical Instrument Digital Interface*) permite la interconexión de distintos instrumentos musicales y accesorios electrónicos juntos (sintetizadores, secuenciadores, máquinas rítmicas y diversos PC). Incluye la definición de un puerto serie específico y un protocolo de transferencia de *órdenes* a través del puerto.

11.4.3 Sintetizadores de voz

Las unidades sintetizadoras de la voz son dispositivos que dan los resultados de un programa emitiendo sonidos (fonemas o palabras) similares al habla humana.

La mayor parte de los dispositivos sintetizadores de voz tienen memorizados digitalmente cada uno de los fonemas o palabras que son capaces de emitir. Los datos que recibe un sintetizador procedentes del procesador o de la memoria (generados, por supuesto por un programa) corresponden a la identificación de los fonemas o palabras a emitir. Una vez que se analiza el dato, se activa una rutina encargada de generar el sonido correspondiente: las muestras binarias de la palabra se llevan a la entrada de un conversor D/A, y la salida de éste, una vez amplificada, a un altavoz.

Los sonidos emitidos resultan muy "metálicos", si no se utilizan algoritmos especiales de enlace entre fonemas. Existen sistemas en los que se programa la entonación o niveles de inflexión de voz, y la velocidad de emisión de fonemas.

Por lo general estos sistemas incluyen programas y rutinas que enriquecen las posibilidades de los mismos, como, por ejemplo, generando frases o combinaciones de palabras. Hay sistemas que "traducen cantidades": el número 425, lo pueden pronunciar como "cuatro, dos, cinco" o como "cuatrocientos veinticinco".

11.4.4 Unidades de reconocimiento del habla

Uno de los campos de investigación actual más relevante relacionado con la Informática es el de reconocimiento del habla. Se pretende una comunicación directa del hombre con el computador, sin necesidad de transcribir la información a través de un teclado u otros soportes intermedios de información.

Desafortunadamente estos sistemas de entrada, que formarían parte de la quinta generación de computadores, no están aún suficientemente desarrollados.

Usualmente los dispositivos de reconocimiento del habla o "de la palabra" (como también se les llama) tratan de identificar fonemas o palabras dentro de un repertorio o vocabulario muy limitado. Un fonema es un sonido simple o unidad del lenguaje hablado. Un sistema capaz de reconocer, supongamos, 7 palabras, lo que hace al detectar un sonido es extraer características o parámetros de dicho sonido, y compararlos con los parámetros de la misma naturaleza, previamente memorizados, de las 7 palabras que es capaz de reconocer. Si, como resultado de la comparación, la palabra a reconocer se identifica como correspondiente a uno

de los 7 patrones (el más parecido), se transmite a la memoria intermedia del dispositivo el código binario identificador de la misma. Si el sonido no se identifica, se indica esta circunstancia al usuario o locutor para que éste vuelva a emitir el sonido.

Un sistema de reconocimiento del habla típico contiene (Figura 11.20):

- **Micrófono**, que convierte las señales sonoras en señales eléctricas.
- **Conversor A/D**, que transforma la señal analógica de entrada en una señal binaria.
- **Circuitos** (o algoritmos) **de preproceso**, que tratan de normalizar la información de entrada, o segmentarla en fonemas o palabras la información de entrada, y de normalizarla. Piénsese que la pronunciación está sujeta a múltiples factores: diferentes personas, distintos tonos, distinta velocidad, etc.
- **Extractor de características**. Cada fonema o palabra de entrada se transforma en un vector de parámetros que caracterizan el elemento de léxico captado.
- El sistema contiene un **diccionario** con los vectores de características de cada uno de los fonemas o palabras posibles (**patrones**). Este diccionario suele crearse en una etapa previa al reconocimiento, denominada **fase de aprendizaje**. Cada patrón del diccionario tiene asociado una etiqueta (código o señal) que es la que hay que dar a la salida del sistema cuando se reconoce un elemento de léxico.
- Las características del elemento de léxico a reconocer se comparan en un **clasificador** con las de los distintos patrones incluidos en el diccionario, y la etiqueta del patrón más parecido se da como señal de salida del sistema.

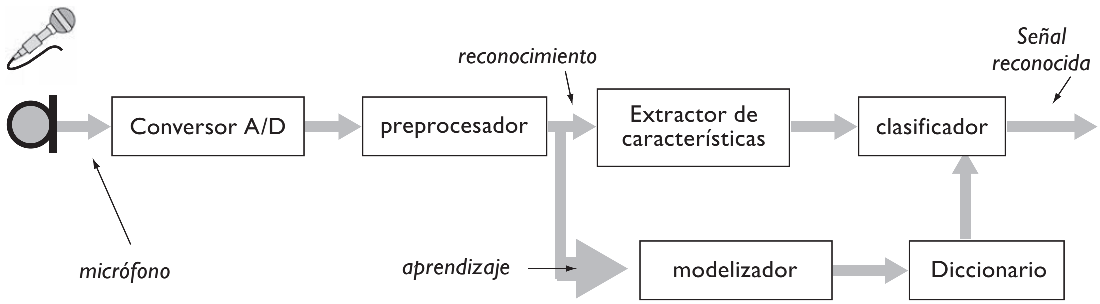


Figura 11.20. *Esquema de un sistema típico de reconocimiento del habla.*

Existen diversos tipos de sistemas de reconocimiento de la voz:

- Dependientes del usuario (o **reconocedor de palabras monolocutor)**. En estos sistemas es necesario realizar la fase de "aprendizaje" o "programación" con el usuario, al cabo de la cual puede reconocer ciertas palabras. En el periodo de aprendizaje el sistema retiene o memoriza las características o peculiaridades de los sonidos emitidos por el locutor, y que luego tendrá que identificar.
- Independientes del usuario (o **reconocedor de palabras multilocutor**). Estos sistemas están más difundidos, pero el vocabulario que reconocen suele ser muy limitado. Los parámetros de las palabras que identifican vienen ya memorizados al adquirir la unidad. Son utilizados, por ejemplo, para dar cifras numéricas, o para definir el movimiento de cierto tipo de robots. En este segundo caso, el operador da verbalmente órdenes elegidas de un repertorio muy limitado, como puede ser: {para, anda, arriba, abajo, izquierda, derecha}. La unidad, cuando capta un sonido, comprueba si corresponde a uno de los seis del repertorio. En caso de identificación se transmite al procesador la

información necesaria para la ejecución de la rutina que pone en marcha y controla la acción requerida.

- **Reconocedores de palabras aisladas**, como su nombre indica pretenden reconocer palabras aisladas. El ejemplo del párrafo anterior es un caso de reconocedor de este tipo.
- **Reconocedores del habla continua**, tratan de reconocer el discurso continuo.

Obviamente los sistemas más complejos son los multilocutor de habla continua.

11.4.5 Unidades para entrada de vídeo

Poco a poco se van introduciendo aplicaciones que requieren la utilización de fotos estáticas e imágenes en movimiento. Numerosas aplicaciones de internet, como la videoconferencia, utilizan estas posibilidades.

Una máquina de fotos digital se puede utilizar para introducir imágenes (fotos) estáticas directamente; es decir, sin necesidad de hacer una foto, revelar el carrete y utilizar un escáner para digitalizar la imagen. Una máquina de fotos digital es una cámara portátil que captura y enfoca, a través de una óptica adecuada, la imagen estática sobre un filtro de colores rojo, verde y azul, la transforma en señales eléctricas por medio de un sensor CCD, la digitaliza con un conversor A/D, y la graba en una memoria PROM (generalmente de tipo *Flash*) o disquete, no requiriendo un carrete fotográfico. Debido a que para cada punto de imagen hay que grabar el color asociado, estas imágenes necesitan una gran capacidad de almacenamiento que se reduce notablemente aplicando técnicas de compresión de datos. La información del disquete puede ser leída por un computador a través de una unidad de disquete convencional. Aunque la calidad de una imagen digital obtenida por una cámara de fotos no es equiparable a una imagen fotoquímica, algunos modelos llegan a obtener resoluciones de más de 3,3 millones de puntos de imagen, más que suficiente para numerosas aplicaciones.

Por otra parte, existen pequeñas cámaras digitales de vídeo que se pueden acoplar a un computador. Estas cámaras tienen una óptica y sistema de sensores (CDC) similar a las cámaras convencionales, pero son más sencillas ya que únicamente tienen que generar imágenes en formato de mapa de bit y transmitirlas al computador; es decir, no incluyen los mecanismos de grabación y tracción para almacenar la imagen en un dispositivo de cinta magnética.

Otra posibilidad de introducir en un computador señales de vídeo es desde una cámara de vídeo convencional y una tarjeta de vídeo, tal y como se indica en la sección siguiente.

11.4.6 Tarjetas de edición de vídeo

Existen **tarjetas de edición de vídeo** que permiten captar señales de radio y/o vídeo, demodularlas y almacenarlas en forma digital, y/u oírlas o verlas, directa o posteriormente, a través del sistema de audio o la pantalla del computador. Básicamente constan de una entrada de antena, un circuito sintonizador y un conversor A/D.

Se suministran con programas que emulan en pantalla los mandos de control de un sintonizador de radio o de TV, y que pueden ser accionados con ayuda del ratón. Estas tarjetas incluyen entradas para las antenas, y también suelen incluir un mando a distancia.

En las tarjetas de edición de vídeo pueden conectarse videograbadoras y cámaras de vídeo convencionales. La información que se recibe, en las formas anteriores o a través de la antena, puede editarse (mezclar con otro tipo de información, superponer imágenes, alterar el orden de las imágenes, etc.) o incluso ser decodificada (caso de canales de TV codificados).

11.5 Entrada/salida de señales analógicas

Según indicamos en la Sección 1.5, en la actualidad existe gran cantidad de computadores de uso específico en forma de **sistemas embebidos**. Estos sistemas digitales contienen los elementos básicos de los computadores y funcionan con determinados programas grabados en su memoria ROM. La naturaleza de los dispositivos de E/S está determinada por la función a realizar por el sistema. Así un computador diseñado para controlar automáticamente el alumbrado de una ciudad ha de tener como entradas, además de un posible teclado para comunicación con el operador, dispositivos físicos que le suministren datos sobre la luz ambiental. También deberá disponer de salidas especiales que, por ejemplo, actúen directamente sobre los interruptores del alumbrado público para encenderlo o apagarlo.

Ejemplos de computadores de uso específico, además del ya indicado, son los siguientes:

- Sistema de control de un proceso industrial.
- Máquina electrónica de juego tragaperras.
- Robot industrial.
- Sistema de control de posición de una antena de radar.
- Máquina herramienta de control numérico.
- Sistema automático de detección y alarma de radioactividad ambiental.
- Sistema de control de tráfico aéreo en un aeropuerto.
- Sistema de autocontrol de vuelo de un misil.
- La mayor parte de los instrumentos electrónicos de medida actuales, incluyendo instrumentación médica (gamma-cámara, tomografía axial computerizada, etc...).

La mayoría de los computadores de uso específico se caracterizan porque deben, además de comunicarse con el operador, captar otros tipos de datos, que son medidas de magnitudes físicas *ambientales* (del entorno). Es más, a veces el computador se diseña de forma que, al conectarla, siempre ejecuta el mismo programa y sólo debe recibir los datos ambientales. Este es el caso, por ejemplo, de una máquina tragaperras. Los datos son la detección de llegada de una moneda y de pulsación de determinados botones o teclas. En otras ocasiones (sistemas de control de procesos, o robot, por ejemplo) el computador contiene unos pocos programas, y el operador da la orden de ejecución de uno u otro accionando determinados pulsadores o interruptores de un panel o consola de control.

Las salidas de estos sistemas también son de naturaleza especial ya que suelen actuar o controlar magnitudes físicas no eléctricas: calentar una habitación, mover una bancada de una máquina herramienta, provocar un desplazamiento del brazo de un robot, etc.

Existen dispositivos que transforman señales físicas de diversa naturaleza (temperatura, presión, posición, etc.) en señales eléctricas, o viceversa. Los dispositivos que generan señales eléctricas son conocidos como **sensores** o **detectores**. Los que transforman una señal eléctrica en otra de distinta naturaleza se denominan **efectores** o **actuadores**. Las señales eléctricas involucradas en estos procesos de transformación se denominan **señales analógicas** ya que varían de forma *análoga* a como la hace la señal física original, pudiendo tomar cualquier valor comprendido dentro de un determinado rango.

En las secciones siguientes vamos a dar unas ideas generales sobre sistemas de entrada y sistemas de salida para señales analógicas; es decir, sistemas que captan o proporcionan datos de tipo analógico.

11.5.1 Sistemas de adquisición de datos analógicos

La mayor parte de las variables físicas de la naturaleza (temperatura, intensidad luminosa, posición, sonido, etc.) son señales o funciones que varían continuamente con el tiempo. Estas señales, con sensores o detectores, pueden convertirse en señales eléctricas analógicas.

Existen sensores específicos para cada tipo de magnitud; así hay detectores de:

- Temperatura (termistores, termopares, etc.).
- Presión.
- Intensidad de luz (fotodetector, etc.).
- Humedad.
- Humo.
- Caudal de líquido.
- Sonido (un micrófono, por ejemplo).
- Nivel de agua.
- Posición lineal (potenciómetros lineales, por ejemplo).
- Posición angular (por ejemplo, potenciómetros circulares, sincrogeneradores, etc.).
- Sensores de señales fisiológicas (que se utilizan en electroencefalografía, o electrocardiología, por ejemplo), etc.

Una vez convertida la señal original en señal eléctrica, se amplifica o atenúa adecuadamente (bloque A de la Figura 11.21). Es necesario a continuación darle una forma apta para ser tratada por el computador, o en otras palabras, ser transformada en datos numéricos binarios según el código que utilice el computador. Esto se hace con unos circuitos electrónicos específicos denominados **conversores analógico/digital** o, abreviadamente, conversores A/D. El conversor A/D capta muestras de la señal analógica de entrada y las mide o digitaliza, dando a su salida un conjunto de bits o número binario que representa el valor de la amplitud de la muestra captada.

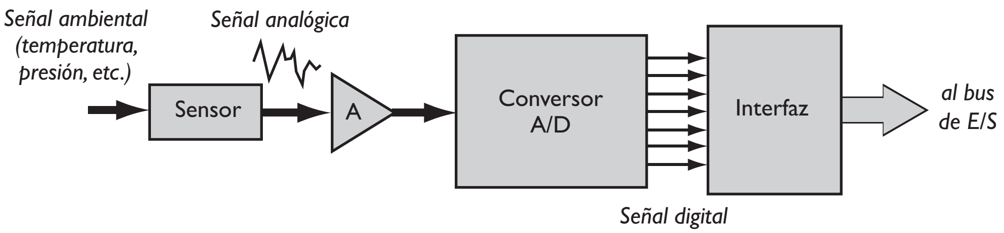


Figura 11.21. *Diagrama de bloques de un sistema de entrada analógica a un computador.*

Dependiendo de la naturaleza de la señal original y de la aplicación del sistema, la velocidad de captación de muestras (velocidad de muestreo) debe ser mayor o menor. Por ejemplo, en el caso del sistema de alumbrado de la ciudad las muestras pueden tomarse con intervalos de 5 minutos o más. El sistema trata de detectar cuándo la intensidad de luz ambiental cae por debajo de un determinado nivel, para, en ese caso, conectar automáticamente el alumbrado público. Entre la toma de una muestra y la sucesiva, el conversor A/D y el resto del sistema podría tomar muestras de distintos barrios de la ciudad, e iluminarlos independientemente. Las distintas muestras analógicas de cada barrio serían convertidas a binario por el mismo conversor. Para conseguir que el mismo conversor se pueda utilizar para distintas señales existe

un circuito denominado **multiplexor analógico** que selecciona uno de entre varios canales de entrada (Figura 11.22). En la figura se selecciona como entrada del conversor uno de los 16 canales de entrada, la selección la realizan las señales de control binarias a, b, c, d. Así abcd = 0000 selecciona el canal 0, abcd = 0001 el canal 1, ..., y abcd = 1111 el canal 15.

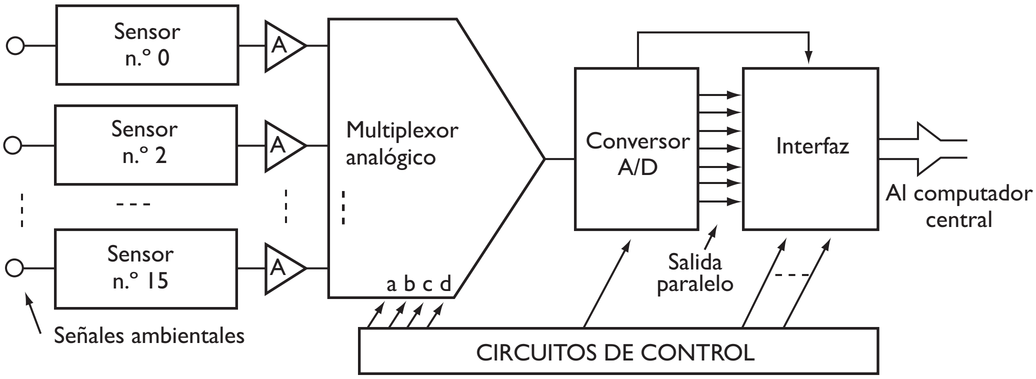


Figura 11.22. *Diagrama de bloques de un sistema de adquisición de datos de 16 canales.*

La Figura 11.22 representa un esquema completo de una unidad de entradas analógicas denominado también **sistema de adquisición de datos analógicos**, en este caso de 16 canales de entrada. Después de los sensores se incluyen unos amplificadores o adaptadores (A) para acoplar las señales analógicas a los rangos de amplitud que requiere el multiplexor analógico. Las señales de control pueden generarse por circuitos especializados incluidos en el periférico (en el sistema de adquisición de datos) o en el propio procesador del sistema, controlándose la generación de las mismas por programa (por software). La conexión con el computador se efectúa como un periférico normal (Capítulo 12).

Comercialmente se dispone de sistemas de adquisición de datos completos (multiplexor, conversores A/D, interfaz, y circuitos de control) en una tarjeta de circuito impreso, incluyendo también sistemas de salida analógica. Estas tarjetas disponen de la interfaz adecuada para conectarse al bus correspondiente del computador.

11.5.1.1 Sistemas de salida analógica

Existen también computadores, fundamentalmente de uso específico, cuya salida debe actuar sobre un sistema o dispositivo controlable por una señal eléctrica analógica. Esta señal actúa sobre un efector generándose así una señal no eléctrica: el cierre de un contacto eléctrico (relé) de gran potencia (para encender unas lámparas o un horno, por ejemplo), o provocar un movimiento determinado excitando un motor (brazo de un robot, o torreta de una máquina herramienta, por ejemplo), abrir una válvula de una conducción de fluido, etc. Los dispositivos físicos más utilizados, controlables por señales eléctricas, son:

- Contactos electromecánicos o relés. En estos dispositivos una pequeña corriente eléctrica en un electroimán provoca, por atracción magnética, el cierre de un interruptor que soporta una gran intensidad (resistencia de un horno eléctrico, etc.). Existen dispositivos electrónicos de semiconductor (tiristores) que realizan la misma función que los relés.
- Electroválvulas. Abren o cierran una conducción de fluido.

- Servomotores. Provocan el giro de un eje.
- Altavoz. Origina la emisión de un sonido.

La conversión eléctrica de un dato binario en una señal analógica se efectúa en un circuito denominado **conversor digital/analógico** o, abreviadamente, conversor D/A. La Figura 11.23 muestra un esquema simplificado de una unidad de salida analógica.

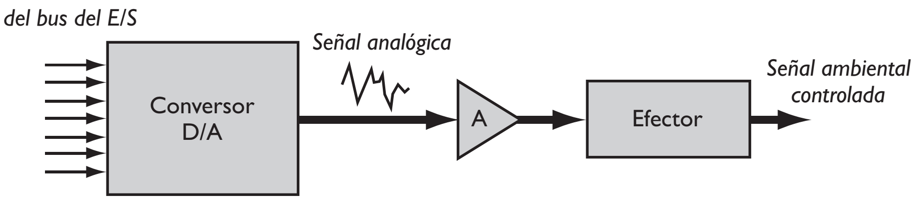


Figura 11.23. *Diagrama de bloques para una salida analógica.*

EJEMPLO 11.15

Un sistema de adquisición de datos[9] *puede tener las siguientes características:*

- *Interfaz USB para conexión con el computador (PCs y equipos portátiles o móviles).*
- *8 canales analógicos de entrada, cada uno de ellos de 12 bits. Admitiendo 4 de ellos muestreo simultáneo.*
- *2 conversores D/A con salidas analógicas (rango de 12 bits y velocidad de 500KS/s, S simboliza muestra, Sample).*
- *8 entradas digitales independientes y 8 salidas digitales independientes.*
- *Rango de la señal de entrada programable.*
- *Memoria FIFO de 4 KS.*
- *Máxima frecuencia de muestreo: 500 KS/segundo.*
- *3 contadores de 16 bits.*

11.6 Conclusión

En este capítulo hemos abordado conceptos básicos sobre los periféricos de E/S de un computador. Estos son los dispositivos para intercambio de información con el exterior.

La descripción de cada uno de los periféricos se ha realizado pretendiendo los siguientes objetivos:

1) Mostrar sus funciones y posibilidades de uso, diferenciando claramente las misiones y prestaciones de cada uno de ellos.
2) Indicar sus principios físicos de funcionamiento.
3) Dar unas nociones generales que faciliten su utilización.

Se ha hecho especial hincapié en las unidades que tienen mayor aplicación en la actualidad o que gozan de cierta importancia histórica o relevancia pedagógica, y no se han analizado otros más avanzados como pueden ser los relacionados con realidad virtual.

[9] USBDAQ-9100MS.

Consideramos haber logrado recopilar y ordenar diverso material técnico de interés que figura muy disperso en distintos libros, manuales de fabricantes, y guías comerciales especializadas. La mayoría de la documentación utilizada se cita en la bibliografía final del capítulo.

EJERCICIOS[10]

11.1 Un escáner de color tiene una resolución de *1.200 dpi* y puede detectar *256* niveles para cada uno de los tres colores básicos. Estimar la memoria necesaria para almacenar una fotografía de 10 x 13 cm.

11.2 En una pantalla de resolución XVGA con 1.280 x 960 píxeles, suponiendo que con un determinado editor cada celda de almacenamiento de caracteres ocupa 36 filas x 25 columnas de píxeles, indicar cuantos caracteres se pueden visualizar por fila y por columna.

11.3 Determinar el tamaño máximo de un píxel y de un punto de pantalla con una resolución Súper VGA en una pantalla de color de 14 pulgadas.

11.4 Obtener el valor de la capacidad de la memoria de cuadro de una pantalla de 25 x 80 celdas, con resolución de color EGA, y atributos especiales de parpadeo, vídeo inverso iluminación tenue y sobreiluminación.

11.5 La propuesta de TV de alta definición requiere una resolución de 1.280 x 1.024 puntos de imagen, con 256 niveles por cada uno de los colores básicos. Determinar la capacidad de la memoria de cuadro que se necesitaría para este sistema.

11.6 En la TV de alta definición se pretende obtener en pantalla una resolución de 1.280 x 1.024 píxeles. Sabiendo que para reproducir correctamente los colores fotográficos se necesitarían 16.777.216 tonalidades diferentes, estimar la capacidad en KBytes de la memoria de imagen que se necesitarían.

11.7 Determinar la resolución, r, que tendría que tener una pantalla de 21 pulgadas para tener una densidad de puntos de imagen de $\lambda = 300$ d/i.

11.8 Una impresora tiene una memoria intermedia de 512 Bytes y escribe a 120 cps. Está conectada a un computador a través de una línea serie asíncrona que transmite a una velocidad de 1.200 bits/s. Teniendo en cuenta que a cada carácter (8 bits) se le añaden 4 bits adicionales de control, hacer una estimación del tiempo que se tarda en rellenar la memoria intermedia. ¿Cada cuánto tiempo tendría que rellenar la memoria intermedia el procesador?

11.9 Una impresora láser puede imprimir hasta 2,5 páginas por minuto (cada página contiene 40 líneas, de 80 caracteres cada una). La impresora está conectada a un computador a través de una línea de transmisión serie asíncrona. La comunicación se realiza añadiendo a cada carácter (8 bits) 3 bits de control para su adecuada transmisión. Determinar la velocidad de transmisión mínima en b/s desde el computador a la impresora para poder mantener la citada velocidad de impresión.

11.10 Se dispone de un adaptador SVGA (1.024 x 768 puntos de imagen, 256 colores) suponiendo que el único atributo de cada punto es el color:
a) Hacer una estimación de la capacidad de la memoria de imagen
b) Con objeto de que la pantalla pueda utilizarse en un sistema multimedia que pretende visualizar imágenes en movimiento (25 Hz), indicar el ancho de banda del bus de datos de la memoria de imagen.

11.11 Suponiendo que con una máquina de fotos digital[11] en un disquete convencional de 3½"(HD) se graban 20 fotografías, la resolución obtenida en cada foto es VGA (640 x 480 puntos de imagen), y la grabación de la imagen se efectúa utilizando el estándar de compresión JPEG de forma que la información codificada representa un 16,38% de la inicial, estimar el número de bits por punto de imagen que se utilizan para codificar el color.

11.12 Una cámara digital de fotos realiza fotografías con una resolución de 2.272 x 1.704 píxeles. Las fotografías se almacenan en una tarjeta

[10] Recordar que 1 pie = 30,48 cm = 12''; 1'' = 2,54 cm.
[11] SONY modelo Mavica MVC-FD5.

de memoria de 128 MB de capacidad usando el sistema de compresión de imágenes JPEG con una relación de compresión de 1 a 8.
a) Después de almacenar en la memoria 92 fotografías nos quedan libres 636,125 KB en la memoria. ¿Cuántos bits de profundidad de color hemos utilizado en cada fotografía?
b) La cámara dispone de una interfaz USB 1.1 para transferir las fotografías a un ordenador. ¿Cuánto tiempo tardará la transferencia de cada una de las fotografías suponiendo que la conexión USB está funcionando a su velocidad máxima (*1,5 MB/s*)?

11.13 En un sistema multimedia se quiere transmitir por un módem una secuencia de imágenes de 320 x 200 píxeles y 256 colores por punto imagen. La transmisión se ha configurado a una velocidad de 33.600 bps y añadiendo un bit de paridad con criterio par por cada byte de información.
a) ¿Cuántas imágenes por segundo se pueden transmitir?
b) ¿Qué velocidad de transmisión se necesitaría si quisiéramos visualizar en el computador receptor 15 imágenes por segundo?
Sugerencia: suponer que no se transmite ninguna información adicional de control.

11.14 En un monitor de 19" queremos obtener una resolución de 1.280 x 1.024 píxeles de imagen. Además para cada uno de ellos tendremos 256 tonalidades distintas para cada uno de los tres colores básicos (rojo, verde y azul). Se desea visualizar en la pantalla una película DVD a 25 imágenes/segundo.
a) Calcular el tamaño mínimo de la memoria de imagen que se necesita.
b) Obtener el ancho de banda mínimo requerido para el controlador de gráficos (AGP).

11.15 En un sistema multimedia se quiere transmitir una secuencia de imágenes de 320x200 puntos imagen (píxeles) y 256 colores por punto imagen por un módem. La transmisión se ha configurado a una velocidad de 33.600 bps (bits por segundo) y añadiendo un bit de paridad con criterio par por cada Byte de información.
a) Cuántas imágenes por segundo se pueden transmitir.
b) Qué velocidad de transmisión se necesitaría si quisiéramos visualizar en el ordenador receptor 15 imágenes por segundo.
Sugerencia: suponer que no se transmite ninguna información adicional de control.

11.16 Para el sistema de adquisición de datos del Ejemplo 11.15:
a) Determinar el ancho de banda entre el sistema de adquisición y el computador.
b) El periodo mínimo de muestreo.
c) Indicar el motivo de por qué dispone de una memoria FIFO, en lugar de una RAM o LIFO.
d) El tiempo que se tardaría en completar la memoria FIFO suponiendo un muestreo continuo.

11.17 Un sistema de adquisición de datos analógicos de 16 canales se utiliza para captar otras tantas señales. Si la frecuencia de muestreo (para no perder información) de cada señal debe ser al menos de f_s = 2 KHz ¿Cuál es el tiempo máximo de conversión posible para el conversor A/D?

BIBLIOGRAFÍA

[Coo95] Cook, B. M.; White, N. H.: *Computer Peripherals,* Edward Arnold, 3ª Edc. 1995.
[Doy99] Doyle, L. F.: *Computer Peripherals,* 2ª Edc. Prentice Hall, 1999.
[Ham03] Hamacher, C.; Vranesic, Z.; Zaky, S.: *Organización de computadores*, McGraw-Hill, 2003.
[Hob00] Hobbs, M.: *Multifuncional peripherals for PCs,* Newnes, 2000.
[Min97] Minasi, M.: *The complete PC upgrade & maintenance guide,* 8ª Edc. Sybex, 1997.
[Mur00] Murdocca, M. J.; Heuring,V. P.: *Principles of Computer Architecture,* Prentice Hall, 2000.
[Nor03] Norton, P.: *Introduction to Computers,* 5th Ed. McGraw-Hill, 2003.

[Pri05] Prieto, A.; Prieto, B.: *Conceptos de Informática,* Serie Shaum, McGraw-Hill, 2005.

[Rag98] Raghavan, S. V.; Tripathi, S. K.: *Networked Multimedia Systems,* Prentice Hall, 1998.

[Ros99] Rosch, W. L.: *Hardware Bible,* 5ª Edc. Que, 1999.

[Sta00] Stallings, W.: *Organización y arquitectura de computadores*. 7ª Edc. Prentice Hall, 2006.

[Tan00] Tanenbaum, A. S.: *Organización de computadoras. Un enfoque estructurado,* 4ª Edc. Pearson Educación, 2000.

[Wil80] Wilkinson, B.; Horrocks, D.: *Computer Peripherals,* Hodder and Stoughton, 1980.

[Woo92] Woodard, O. C.; Long,T.: *Display Technologies,* Byte, págs.159-168, Julio 1992.

[Xer84] XEROX. *The Xerox 2700 Distributed Electronic Printer. System Management Guide,* 600P86/65. Septiembre 83. (Welwyn Hall). Rev. Enero. 1984.

CAPÍTULO 12

Estructura y arquitectura de computadores

El presente capítulo se dedica a ampliar y profundizar algunos de los conceptos vistos en capítulos anteriores sobre procesadores y la estructura de computadores, en general. En primer lugar (Sección 12.1), se describen los conceptos de computadores de repertorio complejo de instrucciones (CISC) y de repertorio sencillo de instrucciones (RISC). Posteriormente (Sección 12.2) se analizan distintos tipos de procesadores integrados para, a continuación (Sección 12.3), citar las principales peculiaridades y aplicaciones de los sistemas embebidos. En los capítulos anteriores se han estudiado los diversos elementos con los que se configura un computador: procesador, memoria interna, memoria externa y periféricos de E/S. En este capítulo completamos esa visión describiendo como se interconectan unos con otros para formar un computador que funcione eficientemente, El problema de la interconexión de las distintas unidades es complejo, ya que cada una de ellas presenta peculiaridades muy diversas, como velocidad de funcionamiento muy divergentes, distintos niveles de tensiones o corrientes para representar los valores lógicos 0 y 1, etc. La Sección 12.4 precisamente describe las estructuras básicas de interconexión. Posteriormente (Sección 12.5) se analizan sucintamente los tipos de buses estandarizados más usados. A continuación (Sección 12.6) tratamos de aplicar los conceptos vistos con anterioridad a un ejemplo concreto, y qué mejor elección, dada su popularidad, que un PC compatible. El capítulo concluye (Sección 12.7) con una sección dedicada al paralelismo en computadores.

12.1 Procesadores CISC y RISC

En esta sección se describen dos enfoques distintos sobre la concepción de los procesadores con respecto a si el repertorio de instrucciones máquina debe ser complejo (procesadores CISC) o no (procesadores RISC).

Según vimos en la Sección 1.3, el tiempo de ejecución de un programa, t_E, es el tiempo que transcurre desde el inicio hasta el final su ejecución. Si denominamos N_I al número de instrucciones que se ejecutan en un programa, N_{CI} al número medio de ciclos de reloj que consume cada instrucción, y F a la frecuencia del reloj, el tiempo de ejecución del programa vendrá dado por:

[12.1]

$$t_E = N_I \cdot N_{CI} \cdot T = \frac{N_I \cdot N_{CI}}{F}$$

Por otra parte, el **rendimiento de un computador** en la ejecución de un programa es la inversa del tiempo de ejecución:

[12.2]

$$\eta_E = \frac{1}{t_E}$$

Hasta mediada la década de los 80 el diseño de los procesadores se realizaba de forma que el repertorio de instrucciones máquina fuese lo más completo posible, ampliándolo sucesivamente con objeto de que los traductores de lenguajes fuesen lo más sencillos posible y, además, con la creencia de que también así la ejecución de los programas sería más rápida por la reducción del número, N_I, de instrucciones a ejecutar del programa. Esta tendencia era seguida por la totalidad de los diseños, incluyendo los computadores DEC VAX y las familias de microprocesadores Intel 80 x 86 y Motorola 680 x 0. Los repertorios de estos procesadores contienen del orden de 200 a 300 instrucciones, muchas de ellas sofisticadas, consumiendo la ejecución de cada una de ellas múltiples ciclos de reloj, N_{CI}. Un procesador que sigue esta tendencia se denomina **procesador CISC** (*Complex Instruction Set Computer*, **computador con repertorio de instrucciones complejo**). No obstante, estudios estadísticos sobre la utilización por los traductores de las distintas instrucciones máquina muestran que muchas de ellas apenas se utilizan, por lo que resultaba aconsejable reconsiderar la idea de que un procesador es más potente y rápido cuanto más complejo es. Surge así la tendencia de **procesadores RISC** (*Reduced Instruction Set Computer*, **computador con repertorio de instrucciones reducido**) cuyas peculiaridades más significativas son:

1. En los repertorios se incluyen tan solo un reducido número de instrucciones que realizan operaciones muy básicas.
2. Los formatos de las instrucciones son muy regulares (igual longitud, igual tamaño y posición dentro de la instrucción de los distintos campos. Código de operación, dirección, etc.).
3. El procesador contiene un número adecuado de registros (de 16 a 64 o más), realizándose las operaciones de la ALU con los datos de dichos registros. Los intercambios de datos entre los registros y la memoria se efectúan sólo con instrucciones específicas de carga en registro (*load*) y de memorizar el contenido de un registro (*store*).
4. Debido a las propiedades anteriores el desarrollo de la unidad de control RISC es mucho más sencillo que en el caso de un CISC, y se logra que el tiempo de diseño y desarrollo de un nuevo procesador se reduzca notablemente, y que pueda ejecutar, por término medio, una instrucción en tan solo un ciclo de reloj.

En resumen, el objetivo de la arquitectura de procesadores es reducir el rendimiento en la ejecución de programas [12.2], cosa que se puede realizar de dos formas [12.1]:

- o bien reduciendo el número de instrucciones máquina del programa (N_I), que es la pretensión de los procesadores CISC,
- o bien reduciendo el número (N_{CI}) de ciclos que utiliza cada instrucción, objetivo de los procesadores RISC.

El problema está en que la reducción el número de instrucciones, N_I , se logra haciendo éstas más complejas, con lo que el número de ciclos por instrucción, N_{CI} , aumenta; y al revés, si hacemos las instrucciones más simples (N_{CI} más pequeño) el número de instrucciones (N_I)

para implementar un algoritmo dado crecerá. En definitiva, para ver la mejora producida hay que considerar conjuntamente los dos parámetros y aplicar la expresión [12.1]. Con técnicas de paralelismo, que se describirán en la Sección 12.7 del presente capítulo, se logra reducir notablemente el valor de N_{CI} sin variar el número de instrucciones del programa con lo que se incrementa notablemente el rendimiento del computador.

La tendencia RISC es utilizada en la actualidad ampliamente en el diseño de muchos microprocesadores (RISC, MIPS, SPARC, PA-RISC, Alpha, PowerPC, etc.), y en realidad muchas de las ideas surgidas con su desarrollo se utilizan también en el diseño los procesadores CISC actuales.

12.2 Procesadores integrados

Desde el desarrollo de los primeros computadores, conforme ha ido evolucionando la tecnología electrónica, se han ido miniaturizando extraordinariamente los elementos constitutivos de las mismos. Así el ENIAC, desarrollado en 1945, ocupaba unos 150 m^2, y un sistema actual tan pequeño como una calculadora programable de bolsillo realiza más funciones que aquél y con precisión y velocidad mucho mayores. Un hito importante en la miniaturización lo logró Intel en 1971 con la comercialización de un circuito integrado al que denominó **microprocesador 4004**. Este primer microprocesador, de palabras de tan solo 4 bits, fue ideado para aplicaciones de control. A partir de entonces diversas empresas fabricantes de circuitos integrados, en las que además de Intel merece la pena citar a Motorola, desarrollaron distintos procesadores integrados. Conviene destacar que aunque los más conocidos por el gran público son los procesadores utilizados para construir computadores de uso general, el mayor número de unidades de microprocesadores se encuentra en muy diversas aplicaciones de electrónica de consumo y, en general, como sistemas embebidos en numerosos sistemas.

Además de los microprocesadores, otros circuitos integrados para procesamiento son los microcontroladores y los procesadores de aplicaciones específicas (Tabla 12.1), a los que se dedicarán las secciones siguientes.

Tabla 12.1. *Tipos de procesadores integrados.*

Microprocesadores	
Microcontroladores	
Aplicaciones especificas:	– Procesadores de señal (DSP, Digital Signal Proccesors) – Procesadores multimedia – Procesadores de red (NP, Networks Proccesors) y de comunicaciones – Procesadores y aceleradores gráficos – Sistemas en un chip (SoC) – etc.

12.2.1 Microprocesadores

Según indicamos en la Sección 1.2.1, un **microprocesador** es un procesador (CPU) implantado en un circuito integrado (*chip*).

Las funciones que realiza un microprocesador son las típicas de un procesador; es decir:

1. Almacena temporalmente las instrucciones.
2. Decodifica los códigos de operación de las instrucciones, y genera las señales de control, tanto para los circuitos internos del propio microprocesador como para los circuitos y dispositivos externos a él.
3. Genera las secuencias de tiempo que sincronizan los intercambios de información entre el microprocesador y su exterior, y que temporizan globalmente al sistema de que forma parte.
4. Contiene un conjunto de registros para el almacenamiento temporal de datos y direcciones.
5. Efectúa las operaciones aritméticas y lógicas típicas de una ALU.

Los microprocesadores más potentes actuales además suelen incluir:

- Procesador de coma flotante (FPU, *Float Point Unit*).
- Unidad de gestión de memoria (*MMU, Memory Management Unit*).
- Por lo general dos niveles (L1 y L2) de memoria caché, permitiendo ampliarla externamente con un nivel adicional (L3). Con frecuencia el nivel L1 está subdividido en dos memorias, una para instrucciones o microoperaciones y otra para datos.

Comercialmente el campo de los microprocesadores ha estado dominado por dos empresas: Intel y Motorola. Los IBM-PC y compatibles, desde sus inicios, 1981, han utilizado microprocesadores de Intel; mientras los Apple y Macintosh han utilizado microprocesadores de Motorola. En la Tabla 12.2 se muestra una lista de los microprocesadores más difundidos y comercializados por Intel incluyendo algunas de sus características más relevantes. Dentro de cada modelo de microprocesador existen distintas versiones con distintas frecuencias de reloj, algunas de las cuales se incluyen en la penúltima columna de la tabla.

La Tabla 12.3 es análoga a la Tabla 12.2 pero está referida a los microprocesadores de Motorola. Puede observarse que las prestaciones de los microprocesadores de Motorola han sido casi siempre superiores (o han ido por delante) de las de Intel; sin embargo, debido a una mejor comercialización y mayor disponibilidad de programas, estos últimos microprocesadores se han utilizado mucho más ampliamente que los primeros.

En 1991 las empresas Apple, IBM y Motorola crearon un consorcio para desarrollar la familia de microprocesadores PowerPC, que son utilizados principalmente por los PC Macintosh fabricados por Apple. El primero PowerPC se construyó con algunos componentes tomados del microprocesador Motorola 68000, pero tomando como base la arquitectura del microprocesador POWER de IBM. La Tabla 12.4 muestra una lista de los principales procesadores de esta serie.

Aparte de Intel y Motorola algunas empresas comercializan otros procesadores con capacidades notables, aunque en algunos casos (como algunos de AMD y Cyrix) tratando de realizar las mismas funciones que los microprocesadores de Intel, Tabla 12.5)

Además de los microprocesadores citados se han comercializado otros muchos, algunos de los cuales han tenido gran éxito en la realización de estaciones de trabajo (Sección 1.5) o sistemas embebidos. En la Tabla 12.5. se incluye un resumen de algunos de ellos.

Para más detalles puede verse la referencia [webMIC], en la web de este libro.

Tabla 12.2. *Microprocesadores más conocidos desarrollados por Intel.*

Generación	Modelo	Fecha	Ancho bus de datos (bits)	Memoria máxima	Caché L1 (interna)	Caché L2 (interna)	Frecuencia reloj	Comentario
4bits: 1970-72	4004	15/11/1971	4	640 B	0	0	0.108 MHz	Primer microprocesador
8bits: 1972-78	8080	1/4/1974	8	64 KB	0	0	2 MHz	
	8085	4/1976	8	64 KB	0	0	3; 5; 6 MHz	
16bits: 1978-85	8086	8/6/1978	16	1 MB	0	0	4 a 12 MHz	Utilizado en IBM-PC-XT
	8088	6/1979	8	1 MB	0	0	5 MHz	Microprocesador del primer IBM-PC
	80286	1/2/1982	16	16 MB	0	0	6-20 MHz	Utilizado en IBM-PC-AT
32 bits: 1989-200xx	80386	17/10/1985	32	4 GB	0	0	16-33 MHz	Se dan las características del modelo DX
	80486	10/4/1989	32	4 GB	8 KB	0	25-100 MHz	Se dan las características del modelo DX. Procesador segmentado. Coprocesador y caché de 8 KB, internos
	Pentium	22/4/1993	64	4 GB	16 KB	0	60-233 MHz	Superescalar: 2 cauces segmentados
	Pentium Pro	1/11/1995	64	64 GB	16 KB	256 KB	133-200 MHz	Puede ejecutar 3 instrucciones a la vez
	Pentium II	7/5/1997	64	4 GB	32 KB	512 KB	266-450 MHz	SIMD con instrucciones MMX
	Pentium II Xeon	1998	64	64 GB	32 KB	512 KB	400-450 MHz	
	Celeron	1998	64	4 GB			266-766 MHz	NMX
	Pentium III	2/1999	64	4 GB	32 KB	256 KB	0,45-1,2 GHz	NMX, SSE
	Pentium III Xeon	1999	64	64 GB	32 KB	1 MB	0,7-3 GHz	NMX, SSE
	Pentium 4	11/2000	64	4 GB	20 KB	256 KB	1,4-1,5 GHz	NMX, SSE, SSE2
	Pentium 4 Xeon	2002	64	64 GB	20 KB	512 KB	1,4-2 GHz	NMX, SSE, SSE2
	Pentium 4 D	2005	64	64 GB		2 x 1 MB	2,80 y 3,20 GHz	Contienen doble núcleo de procesamiento
	Pentium Extreme Edition	2005	64	64 GB	2 x 16 KB	2 x 1 MB	3,20 y 3,46 GHz	Contienen doble núcleo de procesamiento
64 bits: 2001-20xx	Itanium	2001	64	64 GB			0,5-1 GHz	Instrucciones muy largas (VLIW). Caché L3 de 4 MB
	Itanium 2	2002	64	64 GB	32 KB	256 KB	1,0 a 1,7 GHz	Caché L3 de 9 MB
	Itanium 3	¿2006?						Dos núcleos, caché L3 de hasta 24 MB
	Itanium 4	¿2007?						Cuatro núcleos

Tabla 12.3. *Microprocesadores más conocidos desarrollados por Motorola.*

Año	Modelo	Ancho del bus de datos (bits)	Tamaño de registros	Memoria direccionable	Caché interna	Comentario
1973	6800	8	8	64 KB	0	
1979	68000	16	32	16 MB	0	
1983	68010	16	32	16 MB	0	
1984	68020	32	32	4 GB	0	
1987	68030	32	32	4 GB	256 KB	
1989	68040	32	32	4 GB	8KB	Doble de rápido que el 68030
1993	68060	32	32	4 GB	16KB	Último modelo de la serie

Tabla 12.4. *Algunos microprocesadores de la serie PowerPC.*

Denominación Apple	Año	Modelo	Tecno-logía	Frecuencia	Memoria direccionable	Caché interna	Comentario
G1	1992	601	600 nm	33-120 MHz	4 GB	L1 (32KB)	IBM, Apple e Intel inician la serie PowerPC
G2	1993	603	600 nm	300 MHz	4 GB	L1 (32 KB)	Para PC portátiles y sistemas embebidos
	1994	604	500 nm	300 MHz	4 GB	L1 (64KB)	Para PC de sobremesa
G3	1997	750	270 nm	233 y 266 MHz	4 GB	L1 (64KB)	Arquitectura de 32 bits, caché L2 externa
		750 CXr	180 nm	300 a 533	4 GB	L1 (64 KB), L2 (256KB)	Arquitectura de 32 bits
		750 FL	130 nm (SOI)	600 a 733	4 GB	L1 (64KB), L2 (512KB)	
		750 GL	130 nm (SOI)	800 a 933	4 GB	L1 (64KB), L2 (1 MB)	
	2002	750 FX	130 nm (SOI)	600 a 800	4 GB	L1 (64KB), L2 (512KB)	
		750 GX	130 nm (SOI)	800 MHz a 1 GHz	4 GB	L1 (64KB), L2 (1 MB)	
G4	1999	G4 (74xx)	150 nm	350 MHz a 1 GHz	4 GB	L1 (64KB), L2 (1 MB)	ALU de 64 bits. Capacidad SMP, Caché L2 en chip
G5	2001	G5 (970 FX)	90 nm (SOI)	1,4 a 2,2 GHz	8 GB	L1 (86KB), L2 (512 KB)	Arquitectura de 64 bits, 58 millones de transistores. Un núcleo Vectorial/SIMD
	2005	G5 (970 MP)	90 nm (SOI)	1,4 a 2,5 GHz	8 GB	L1 (86KB), L2 (1 MB)	Dos núcleos

G1 a G5:	Procesadores utilizados por Apple para construir sus PC Macintosh
6xx, 7xx, 7xxx	Procesadores de altas prestaciones para PC de sobremesa y servidores. Sistemas embebidos de altas prestaciones.
8xx, 8xxx	Procesadores para comunicaciones y electrónica de consumo.
5xx, 5xxx:	Procesadores para comunicaciones móviles
750CXr, FL y GL:	Electrónica de consumo, redes, sistemas embebidos y aplicaciones de bajo consumo (portátiles)
970:	Computación de altas prestaciones, electrónica de consumo, redes, estaciones de trabajo y servidores.

Fuente: *[Ort05] y http://www-03.ibm.com/chips/power/powerpc/.*

Tabla 12.5. *Otros microprocesadores de interés.*

Fabricante	Modelo	Año	Frecuencia de reloj (MHz)
Advanced RISC Machines	ARM7TDMI		80-133
	ARM9TDMI		200
	ARM10TDMI		300-400
	StrongARM SA-110		160-200
	Serie AMD K5	1995	90-133
AMD	Serie AMD K6	1997-2001	166-570
	Serie AMD K7	1999	500-2.333
	Serie AMD K8	2003	1.400-2.800
DEC	Alpha 21064	1992	200
	Alpha 21164	1996	300
	Alpha 21264	1998	525
	Alpha 21364		1.000
HP	PA7100	1992	
	PA7200	1994	
	PA8000	1996	180
	PA8500	1998	400
IBM	Power1	1990	
	Power2	1993	66,7
	Power3	1997	200
	Power4	2001	< 1.300
	Power5	2005	2.000
MIPS Comp. Systems	R2000	1985	16,7
	R3000	1988	25
MIPS Technology	R4000	1991	100
	R4400	1992	150
	R4600	1994	133
	R5000	1996	180
	R8000	1994	90
	R10000	1995	200
	R12000	1998	300
	R14000	2001	600
	R16000	2002	700
	R16000A	2004	800
SUN	Sparc V7	1987	16,6
	SúperSparc	1991	60
	SúperSparcII	1995	90
	UltraSparcII	1996	250

12.2.2 Microcontroladores

Un **microcontrolador** es un circuito integrado que contiene, total o parcialmente, los cinco elementos básicos de un computador completo (unidad de control, camino de datos y puertos de entrada/salida), estando proyectados para aplicaciones de supervisión, monitorización, gestión y control en sistemas tales como aparatos telefónicos, electrodomésticos, instrumentación médica, control de robots, líneas de ensamblado, etc. Se diferencian de los microprocesadores en que:

1. Contienen en su interior no sólo el procesador, sino también otros elementos como puertos de entrada/salida y memoria principal (ampliable externamente).
2. Están orientados a aplicaciones específicas de control.

EJEMPLO 12.1

El circuito integrado 8051 dispone de 40 terminales (patillas) de entrada/salida, una CPU de 12 MHz, memoria ROM de 4 KB, y memoria RAM de 128 Bytes, y 32 líneas de E/S. Usualmente estos circuitos se utilizan para aplicaciones tales como control de semáforos, control en máquinas lavadoras, etc.

Los microcontroladores están ideados para interactuar con su entorno, captando señales de medida y generando señales de control. Por lo general un microcontrolador consta de los siguientes componentes (Figura 12.1):

- Procesador (núcleo) (UT + UC).
- Memoria no volátil para contener el programa: de tipo EEPROM o, mejor, de tipo Flash.
- Memoria RAM para grabación y lectura de datos.
- Controladores de buses y puertos de E/S:
  - Serie (USB, por ejemplo).
  - Paralelo.
  - Red inalámbrica, etc.
- Elementos y funciones adicionales:
  - Reloj interno.
  - Conversores analógico/digital (A/D) y digital/analógico (D/A).
  - Temporizadores activados por interrupciones para programar distintos intervalos de tiempo.
  - Comparadores analógicos, que detectan umbrales programables de señales analógicas externas.
  - Sistema para entrada y salida de un estado de bajo consumo (reposo o hibernación).
  - Perro guardián (watchdog), que permite reinicializar automáticamente el sistema si éste se bloquea. Es un dispositivo formado por un contador descendente que puede ser utilizado para recuperar el control del microcontrolador cuando se produce un bloqueo o perturbación en el software.
  - Protección ante fallos de alimentación.

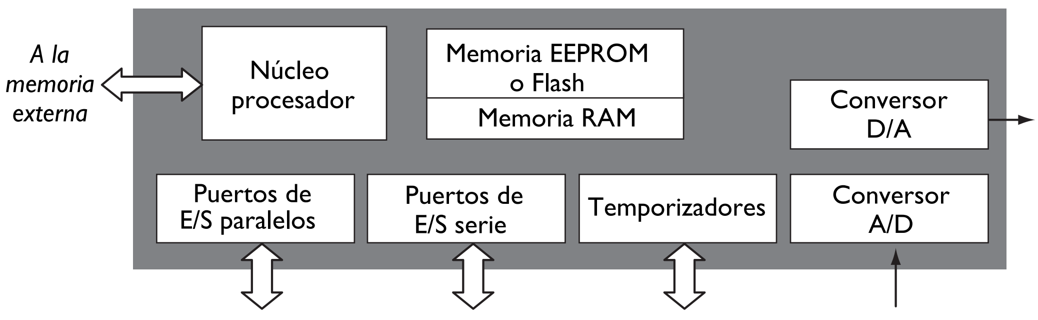


Figura 12.1. *Esquema simplificado de un microcontrolador.*

12.2.3 Otros procesadores integrados

Aparte de los microprocesadores y microcontroladores, se han desarrollado otros procesadores cuyo diseño ha sido proyectado para campos de aplicaciones más concretos (Tabla 12.1). Entre otros, cabe destacar los siguientes:

- **Procesadores digitales de señal** (**DSP**, *Digital Signal Proccesors*), frecuentemente contienen conversores A/D y D/A, y disponen de instrucciones máquina especializadas en facilitar la realización de funciones típicas de procesamiento de señales digitales: Filtrado, Transformada Rápida de Fourier, etc. Se utilizan para aplicaciones tales como procesamiento del habla, control de tráfico de voz a través de redes de conmutación de paquetes (ATM o internet), compresores/decompresores (audio MP3, vídeo MPEG4), Codificadores/decodificadores de audio/vídeo, infraestructura para comunicaciones inalámbricas, etc. Ejemplo, familia TMS320.
- **Procesadores de red** (**NP**, *Networks Proccesors*) y de comunicaciones. Están diseñados para dispositivos de red, tales como enrutadores (routers): direccionamiento de paquetes de datos (análisis de cabecera, etc.). Realizan muy eficientemente procesamiento de protocolos y transferencias de datos Muy similares a los DSP, con unidades segmentadas para aplicaciones específicas. Ejemplo, controlador de red local de Ethernet AM7990.
- **Procesadores y aceleradores gráficos**. La visualización de una imagen en la pantalla de un computador es muy compleja. En efecto, es necesario, a partir de un modelo de la imagen, normalmente constituido por una descripción vectorial de cada uno de los objetos gráficos o primitivas que la forman (que incluye su posición, escala y otros atributos) en elementos bidimensionales a convertir en píxeles de pantalla. Para realizar estas operaciones con calidad y rapidez se utilizan procesadores especializados, como, por ejemplo, el acelerador multimedia *n*Vidia NV1 o el acelerador gráfico Fujitsu TGPx4. Para más detalles puede consultarse la referencia [Mig01].
- **Sistemas en un chip** (**SoC,** *Syntems on a Chip*). Se han ideado un conjunto de técnicas que permiten diseñar y construir o particularizar, con relativa facilidad, un chip para una aplicación concreta, conteniendo en su interior la práctica totalidad de elementos electrónicos de un sistema completo. Constituyen una alternativa al diseño con microprocesadores, microcontroladores y DSP, permitiendo además incluir funciones adicionales. Su campo de aplicación se centra en la construcción de sistemas embebidos. Para el desarrollo de un SoC se utilizan metodologías de co-diseño hardware/software, que permiten establecer la forma más adecuada de implementar una función determinada de un sistema (si en forma de circuito o de programa), y como componente físico se suelen utilizar **Dispositivos lógicos configurables por el usuario** (**FPLD**, *Field Programmable Logic Devices*) que son circuitos integrados en cuyo interior se incluye gran cantidad de módulos lógicos y de memoria cuyo interconexionado puede ser establecido con relativa facilidad incluso en empresas o departamentos especializados de pequeño tamaño. En las FPGA se pueden incluir **módulos IP** (*Intelectual Property*), que son descripciones de elementos de complejidad muy diversa, desde funciones específicas (módulos aritméticos, FFT, etc.) hasta procesadores completos (PowerPC, ARM, etc.).

12.3 Sistemas embebidos

Los microprocesadores y microcontroladores son circuitos que, por ser muy versátiles, pueden construirse en grandes series a muy bajo precio. Además de utilizarse para construir

computadores se recurre a ellos para construir sistemas embebidos en multitud de aparatos, de hecho la mayor parte de los sistemas electrónicos digitales actuales se construyen con procesadores integrados, por ser una opción muy económica. Un **sistema embebido** es un sistema controlado por un procesador integrado cuyo hardware y software están especialmente diseñados y optimizados para resolver un problema concreto. Por lo general estos sistemas interactúan continuamente con su entorno con objeto de monitorizar o controlar algún proceso. Su hardware usualmente se diseña utilizando circuitos integrados comerciales y la mínima circuitería adicional requerida para la aplicación concreta. El término embebido se refiere al hecho de que la electrónica (procesador integrado y otra circuitería) se encuentran incluidos (embebidos o empotrados) en el interior del sistema que controlan, sin ser externamente aparente su existencia. Hay sistemas embebidos (Tabla 12.6) en multitud de electrodomésticos (lavadoras de ropa, lavavajillas, hornos de microondas, etc.), equipos musicales y de vídeo (lectoras/grabadoras de casetes, CD, televisores, vídeos, máquinas de fotos, etc.), juguetes electrónicos (consola de videojuegos), sistemas de control industrial, terminales punto de ventas, comunicaciones de datos (teléfonos inalámbricos, teléfonos móviles, módem, etc.), equipos militares y armamento, etc.

Tabla 12.6. *Campos de aplicación de los procesadores integrados como sistemas embebidos.*

Audio	Radio digital, sistemas de amplificación, compresores MP3, aparatos musicales.
Vídeo e Imágenes	Reproductores de CD y de DVD, cámaras, videoconsola, compresión de imágenes (MPEG4), codificadores/decodificadores.
Computadores	Controladores en impresoras, controlador de teclado, controlador de ratón, escáner, fotocopiadora, unidad de disco.
Redes y telecomunicaciones	Modems, teléfonos digitales, teléfonos celulares, comunicaciones inalámbricas, fax, cajeros, transmisión de voz sobre paquetes (V o P) a través de líneas digitales (IP o ATM), conmutadores multiservicio, pasarelas, comunicación a través de redes inalámbricas, mando de garaje, etc.
Electrodomésticos	Lavadoras, hornos, lavavajillas, secadora, aspiradora.
Industrial	Control de motores, control de temperatura, climatizador, compresor, control de procesos, electromédica, instrumentación de medida, sistemas de seguridad.
Automóvil	Control de encendido, frenado ABS, control de seguridad (bolsa de aire, etc), control de crucero, climatización sistema de mantenimiento, inyección de combustible, instrumentación, GPS, etc.
Edificios inteligentes y domótica	Control de vigilancia y seguridad, control de energía, comunicaciones, confort y salud medio-ambiental.
Aplicaciones militares	

También los computadores de uso general, además de su microprocesador central, contienen diversos procesadores integrados en forma de sistemas embebidos que están, por ejemplo, en el teclado, el monitor (acelerador de gráficos), controlador DMA, controlador de red local y de red WiFi, y en los distintos periféricos.

A diferencia de un computador de uso general, un sistema embebido no es programable por el usuario final de la aplicación. El programa o programas (generalmente grabados en memoria ROM) forman parte integral del sistema y usualmente se ejecuta bajo el control de un sistema operativo extremadamente sencillo.

EJEMPLO 12.2

Un automóvil moderno puede contener del orden de 100 microprocesadores o microcontroladores embebidos que controlan funciones tales como:

- *Control GPS.*
- *Climatización.*
- *Encendido.*
- *Desplazamiento de la transmisión.*
- *Control de potencia.*
- *Antibloqueo del freno.*
- *Control de tracción.*
- *Seguridad (disparo de la bolsa de aire, etc.).*

EJEMPLO 12.3

Uno de los numerosos sistemas que utiliza un procesador integrado (microcontrolador) es un aparato de aire acondicionado (climatizador). En esencia un climatizador se compone de tres elementos básicos: unidad exterior (compresor), unidad interior (consola) y telemando que se comunica con la consola a través de infrarrojos. Básicamente el usuario, por medio del telemando, puede dar las siguientes órdenes de entrada:

- *Conectar/Desconectar.*
- *Temperatura de consigna, T_c, que es la temperatura ambiental que desea.*
- *Selección de frío o calor o sólo ventilación.*
- *Velocidad del ventilador, que, por ejemplo, puede regular en tres valores.*
- *Posicionamiento manual de los deflectores de aire, para que éste se dirija hacia una dirección determinada.*
- *Movimiento constante de los deflectores de aire.*

Además el climatizador recibe como entrada la temperatura ambiente, T_a.

Las salidas del sistema pueden ser las siguientes:

- *Actuación sobre el motor de ventilación:*
  - *v1v0 = 00 (parado), v1v0 = 01 (velocidad mínima)*
  - *v1v0 = 11 (velocidad media), v1v0 = 10 (velocidad máxima)*
- *Actuación sobre el compresor:*
  - *c1c0 = 00 (parado), c1c0 = 01 (arranque)*
  - *c1c0 = 11 (frío en la consola), c1c0 = 01 (calor en la consola)*
- *Actuación sobre las paletas del difusor de aire:*
  - *d2d1d0 = 000 (posición 0, aire a la derecha), ...*
  - *d2d1d0 = 111 (posición 15, aire a la izquierda)*
- *Pilotos indicadores, mostrando el estado de funcionamiento del climatizador (apagado, encendido, etc.).*
- *Visualizador del telemando.*

EJEMPLO 12.3 (Cont.)

Las acciones de control del climatizador requieren unas operaciones muy sencillas que pueden ser realizadas por un microcontrolador (Figura 12.2). Así, dichas acciones son del tipo siguiente:

- *Si el climatizador está conectado y Tc = Ta; entonces hacer: v1v0 = 00 (parar ventilador) y c1c0=00 (parar compresor).*
- *Si el climatizador está conectado, el compresor parado y se requiere una acción de este último, ponerlo en estado de arranque (c1c0 = 01) durante un minuto.*
- *Si el climatizador está conectado y Tc > Ta; entonces hacer: = 01 (producir calor en la consola), y v1v0 = 01 (velocidad baja):*
  - *Si Tc > 1,4 Ta poner el ventilador a su velocidad máxima (v1v0 = 11)*
  - *Si Tc > 1,2 Ta poner el ventilador a su velocidad media (v1v0 = 10)*
- *Si el climatizador está conectado y Tc > Ta; entonces hacer: = 11 (producir frío en la consola), y v1v0 = 01 (velocidad baja):*
  - *Si Tc < 1,4 Ta poner el ventilador a su velocidad máxima (v1v0 = 11)*
  - *Si Tc < 1,2 Ta poner el ventilador a su velocidad media (v1v0 = 10)*
- *etc.*

Las distintas variables se pueden asociar a bits de los puertos de entrada o salida, según sean aquéllas de entrada o salida, respectivamente. La acción de temporización (1 minuto) se puede realizar con ayuda de alguno de los controladores internos del microcontrolador que se utilice. Los programas se grabarían en la memoria EEPROM del microcontrolador.

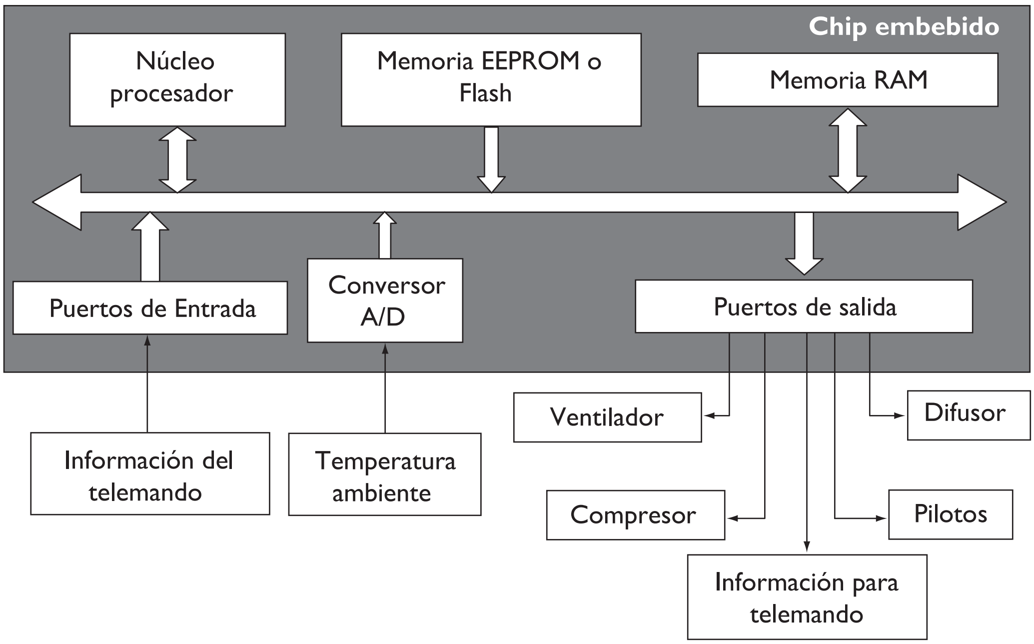


Figura 12.2. *Esquema simplificado del sistema de control de un climatizador.*

Los procesadores integrados han hecho posible que los computadores (de uso específico) se encuentren disponibles en la mayoría de los sistemas electrónicos de consumo que utilizamos cotidianamente. Según se muestra en la Tabla 12.7, podemos establecer cuatro eras de computadores, atendiendo a la disponibilidad de los mismos.

Tabla 12.7. *Eras de computadores según su disponibilidad.*

• Grandes computadores (uno por empresa); 1950.
• Minicomputadores (uno por departamento); 1970.
• PCs (uno por persona); 1980.
• Embebidos (varios por persona, en múltiples sistemas): 2000.

Más detalles sobre sistemas embebidos, pueden verse, por ejemplo, en el Capítulo 9 de la referencia [Ham03].

12.4 Estructuras básicas de interconexión

Las unidades funcionales de un computador se interconectan de acuerdo con una determinada organización. Hay diferentes posibilidades, las más relevantes de las cuales se describen a continuación.

Las formas más sencillas de interconexión de unidades son las mostradas en la Figura 12.3. La primera forma (Figura 12.3a) dispone de un bus específico de interconexión procesador-memoria. Obsérvese que todo el tráfico de información entre periféricos y memoria necesariamente ha de hacerse a través del procesador. La segunda configuración (Figura 12.3b) es más sencilla, y utiliza un único bus (**estructura unibús**). Todas las unidades se conectan a él, y coincide con la estructura del computador CODE-2 analizado en los Capítulos 8 y 9. Este bus se denomina **bus del sistema**. Como en un instante dado sólo puede transmitirse una información por el bus, sólo una unidad (el procesador) puede tener el control del bus del sistema. La ventaja principal de esta estructura es su bajo coste y su flexibilidad para conectar periféricos.

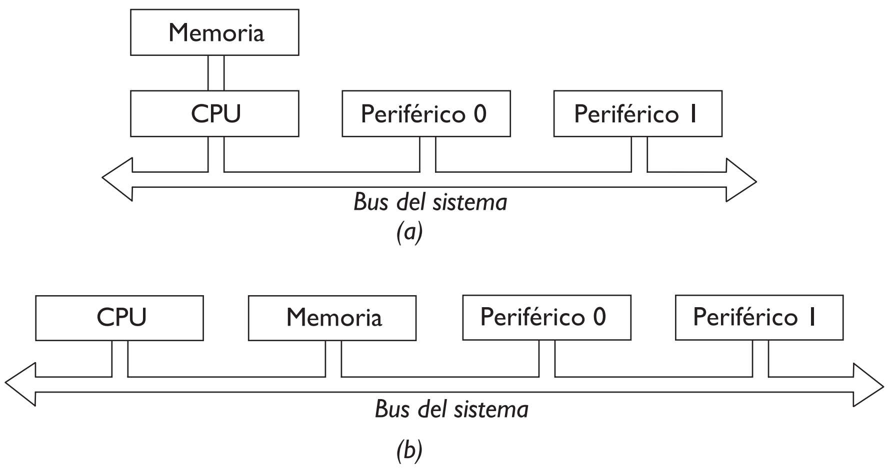


Figura 12.3. *El Organizaciones sencillas de un computador: (a) con un bus específico para conexión procesador memoria, y (b) estructura de bus único.*

El esquema unibús tiene el gran inconveniente de que, en toda transferencia a través del bus, el elemento más lento es el que impone la velocidad de transmisión. Para reducir los efectos de la gran diferencia de velocidades entre el procesador y los periféricos, cada uno ellos contiene una **memoria intermedia** o *buffer*, que almacena la información durante la transferencia. El procesador puede cargar el *buffer* a alta velocidad, y el periférico, a su ritmo, grabar (en el caso de una unidad de disco) o imprimir (en el caso de una impresora) la información existente en el *buffer*. Soluciones complementarias son el uso de controladores de E/S y de controladores de acceso directo a memoria.

Un **controlador** (o **canal** o **procesador**) **de entradas/salidas** (o de **periféricos**) es un procesador, con un amplio *buffer* y especializado en controlar las operaciones de transferencia de datos entre los periféricos conectados a él, y entre estos y el procesador (Figura 12.4). Estas operaciones de control en principio son tarea del procesador, por lo que los controladores de E/S descargan a éste de las mismas: el procesador puede seguir trabajando con la memoria mientras los periféricos concluyen sus operaciones. No obstante, siempre es el procesador quién inicializa y cede el control al controlador, *programándolo* (en un puerto del controlador escribe una palabra de control indicando el tipo de transferencia –lectura/escritura–, los periféricos involucrados en la transferencia y el tamaño del bloque de datos). Una vez que un controlador de E/S finaliza la operación encomendada por el procesador, envía a éste una señal de interrupción indicando que está listo para realizar otra operación. Cada controlador de entradas/salidas monitoriza la actuación de distintos periféricos (Figura 12.4) y su conexión con el bus del sistema suele denominarse **canal de entrada/salida** o sencillamente **canal**. En la Figura 12.4a se muestra una estructura con un bus único que incluye memoria caché y controladores de E/S. Esta estructura de bus único (como la de la Figura 12.3b) tiene el inconveniente de que una orden de transferencia de datos entre periféricos de dos canales distintos puede interferir con, por ejemplo, la captación de una instrucción de la memoria principal por el procesador. Para solventar este problema usualmente se utiliza una estructura con dos buses, tal y como se muestra en la Figura 12.4b, en la que aparece un bus específico de E/S.

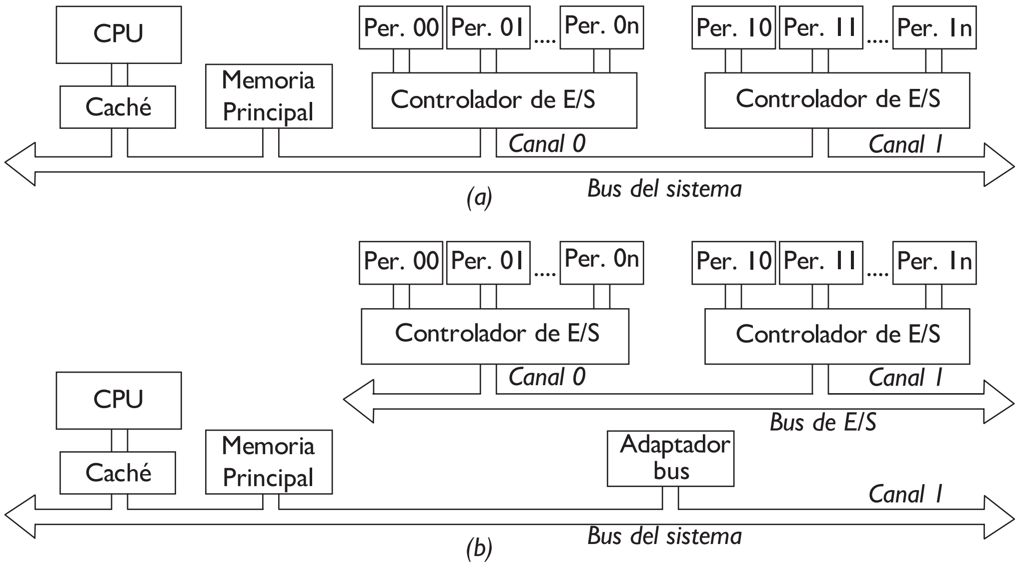


Figura 12.4. *Configuraciones con controladores de E/S: (a) de bus único; (b) con un bus específico de E/S.*

Los controladores de E/S evitan que el procesador tenga que adaptarse entre operaciones individuales de E/S a la velocidad y ritmo que marca el periférico con el que se hace la transferencia. No obstante, el procesador sigue siendo responsable de dichas operaciones. La mayoría de las operaciones de E/S se hacen entre la memoria y un periférico, y viceversa; como ocurre, por ejemplo, en la carga en memoria de un programa para ejecutarlo, o al salvar en disco un documento generado con ayuda de un procesador de textos. Pero, como el control del bus principal del sistema lo lleva el procesador, y las instrucciones de E/S de los lenguajes máquina comúnmente realizan estas operaciones entre el periférico y uno de los registros del procesador, en cada operación elemental de E/S debe intervenir el procesador.

EJEMPLO 12.4

Por ejemplo, la transferencia de un bloque de 1.024 palabras de disco a memoria, necesitará un mínimo de 1.024 cargas en un registro del procesador y 1.024 almacenamientos en memoria, sucesivos. En el caso de CODE-2, supóngase que el puerto de entrada de datos asociado a una unidad de disco fuese IP(4), y utilizásemos el registro rC del procesador para realizar la transferencia, habría que ejecutar las dos siguientes instrucciones tantas veces como datos hubiese (suponemos que en r1 inicialmente hay almacenado 0001):

```
IN rC, IP4          ; llevar dato de disco a procesador (registro rC)
ST v, rC            ; almacenar en M(v + rD) el dato
ADDS rD, rD, r1     ; incrementar en 1 el puntero de memoria
```

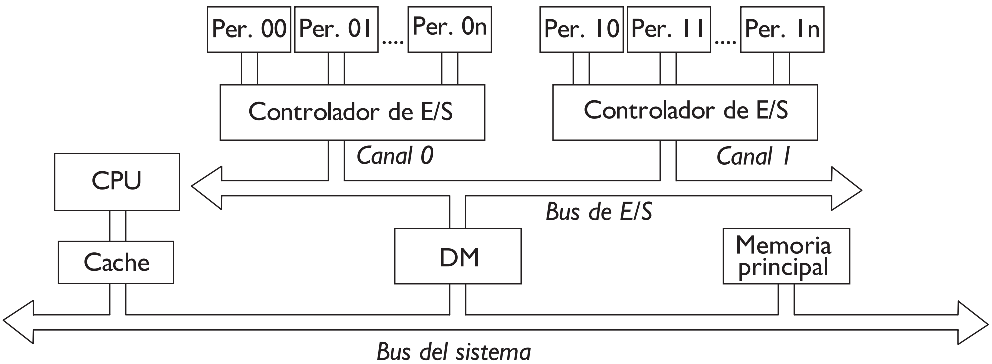


Figura 12.5. *Estructura de un computador con DMA y canal específico de E/S.*

Para descargar al procesador de este trabajo se han ideado los **controladores DMA** (*Direct Memory Access*, **acceso directo a memoria**) (Figura 12.5). El controlador DMA (**DMAC**) es un procesador especializado que permite transferir datos entre memoria y un periférico (y viceversa) mientras el procesador puede realizar otras tareas. El DMAC es externo al procesador y actúa, combinadamente con el procesador, como controlador del bus. Cuando se hace una transferencia de un bloque de información controlado por DMA, previamente el procesador (a través del bus del sistema) programa al controlador inicializando un puerto de control del DMAC en el que se especifica: el tamaño del bloque a transferir, la dirección inicial del bloque a leer en la unidad de origen, la dirección inicial donde se escribirá el bloque en la unidad de destino, y el sentido de la transferencia (lectura o escritura). Una vez inicializado el DMAC, el

procesador cede el control del bus al DMAC, encargándose éste de controlar la transferencia del bloque de información. Mientras tanto el procesador puede realizar otras tareas. Una vez finalizado el trasvase del bloque, el DMAC envía a través del sub-bus de control una señal de interrupción al procesador, para que éste vuelva a tomar el control del bus del sistema. En la Sección 9.5.4.3 se estudiaron más detalles sobre el funcionamiento del DMA.

Existen distintos buses normalizados, los más conocidos se describen en la siguiente sección.

12.5 Buses

Según indicamos en las Secciones 1.2 y 7.1 los distintos elementos de un computador se interconectan por medio de buses, que proporcionan un camino de comunicación para el flujo de datos, direcciones y señales de control/estado entre los distintos elementos. Se presenta el problema de que cuando se comunican dos elementos a través de un bus, el más lento establece la velocidad de transmisión, y para solucionarlo y obtener un buen rendimiento en el funcionamiento global del computador resulta lógico que:

- Se utilicen distintos buses en un mismo sistema, pudiéndose así realizar transmisiones simultáneamente entre distintos elementos a través de buses diferentes.
- Se establezca una jerarquía de buses, en función fundamentalmente de la velocidad.
- Cuando la velocidad de transferencia a un periférico no es elevada se utilice una conexión serie (transmisión bit a bit), y cuando no sea así se utilice una conexión paralelo (varios hilos conductores que transmiten simultáneamente 8, 16, 32, 64 ó 128 bits, dependiendo del tipo de bus).

Los circuitos integrados (chips) de cada subsistema se interconectan por medio de **tarjetas de circuito impreso** o **PCB** (*Printed Circuit Board*). Estas tarjetas son rectangulares, de un material aislante rígido y plano, y contienen las interconexiones eléctricas realizadas por medio de líneas (pistas) metálicas (usualmente de cobre) que van trazadas en las dos superficies de la tarjeta y en capas internas paralelas a su superficie. La tarjeta tiene orificios donde se insertan los distintos circuitos integrados, y regletas de conectores para la interconexión de otras tarjetas o elementos externos. Las conexiones eléctricas de los distintos elementos incluidos se realizan con soldaduras de estaño a las pistas de conducción (buses) correspondientes. En las regletas se insertan otras tarjetas (**tarjetas de expansión**) con las que la conexión eléctrica se hace por presión, de aquí que los contactos de cada regleta y de la tarjeta insertada en ella suela ser de un material conductor de muy baja resistividad (aleaciones con cobre o plata o, incluso oro). En los equipos grandes las tarjetas se instalan en un chasis conectándose una con otras por la parte posterior del mismo (panel trasero).

En general hoy día suelen considerarse los tipos de buses que se indican en la Tabla 12.8.

Tabla 12.8. *Jerarquía de buses en un computador.*

• Buses internos a los circuitos integrados.
• Bus delantero (FSB: *Front Side Bus*): une el procesador con la memoria y el chipset.
• Buses locales para interconexión de elementos de una PCB.
• Bus del panel posterior: conexión entre las PCB dentro de un mismo chasis.
• Buses de expansión, para interconexión de subsistemas (unos 10 m).
• Buses de entrada/salida, para periféricos { – Paralelo – Serie }

Tabla 12.9. *Comparación de características de varios buses.*

Nombre	Anchura de datos	Velocidad máxima de transferencia
ISA (Bus-PC)	8 bits	< 1,2 MB/s
ISA-AT	16 bits	5,33 MB/s
EISA	32 bits	33 MB/s
VLB	32, 64 bits	160 MB/s ; 250 MB/s
PCI	32, 64 bits	132 MB/s ; 264 MB/s
PCI 2.0	32, 64 bits	264 MB/s ; 528 MB/s
PCI-X 2.0	32, 64 bits	2,1 MB/s ; 4,2 GB/s
QuickRing	32 bits	350 MB/s
VME	16 a 23 bits	27,9 MB/s
Multibus II	32 bits	40 MB/s
Futurebus+	64, 128, 256 bits	95,2 MB/s, 3,2 GB/s
SCSI	8 bits	5 MB/s
SCSI-2	8/16 bits	10 MB/s / 20 MB/s
Ultra SCSI	8/16 bits	20 MB/s / 40 MB/s
Ultra640 SCSI	16 bits	640 MB/s
ATA-1 (IDE)	16 bits	3,3; 5,2; 8,3 MB/s
ATA-2/3 (EIDE)	16 bits	13,3 ; 16,7 MB/s
ATA-7 (Ultra ATA-133)	16 bits	133 MB/s
Port AGP 1x	32 bits	264 MB/s
Port AGP 12x	32 bits	2,98 GB/s
FireWire (IEEE 1394)	1 bit *(serie)*	96; 192 y 384 Mbits/s
FireWire (IEEE 1394b)	1 bit *(serie)*	800 Mbits/s; 1,6 Gbits/s y 3,2 Gbits/s
USB 1.0	1 bit *(serie)*	1,5 Mbits/s ; 12 Mbits/s
USB 2.0	1 bit *(serie)*	1,5; 12 y 480 Mbits/s
PCI-Express	1 bit *(serie)*	128 Gbits/s

Unos buses con otros se interconectan por medio de adaptadores o puentes; éstos pueden ralentizar el funcionamiento de los periféricos, de manera que a la hora de configurar un sistema es necesario hacer un estudio de velocidades tanto de los distintos subsistemas que lo integran, como de la transferencia de información en los buses.

Existen definiciones normalizadas o estandarizadas de buses, cuyas especificaciones puede utilizar el ingeniero a la hora de diseñar un computador. Estas normalizaciones especifican cuestiones en los siguientes niveles:

- **Nivel mecánico:** soporte (chasis, circuito impreso, etc.), número de líneas, tipos de conectores, dimensiones de las tarjetas, etc.
- **Nivel eléctrico:** alimentación eléctrica, impedancias, niveles de las señales de tensión o de corriente, etc.
- **Nivel lógico:** número de señales, niveles de tensión o corriente correspondientes al 0 ó 1 lógico, etc.
- **Nivel de temporización:** intervalos de tiempo correspondientes a cuando los datos son válidos, direcciones y señales de control, etc.
- **Niveles de transferencias:** protocolos de arbitraje, detección de errores, etc.

Algunas de las normalizaciones de buses más conocidos son (Tabla 12.9):

Bus PC/XT
Fue introducido en 1981 con el IBM-PC y define 62 hilos, constituidos por 3 de tierra, 5 de tensiones de alimentación, 20 líneas de dirección, 8 líneas de datos, 10 líneas para señales de interrupción, y una gran variedad de líneas para usos específicos. Es un bus estrictamente para arquitecturas de 8 bits (esto es, datos de 8 bits).

ISA Bus
Fue introducido en 1984 con los IBM-PC AT (microprocesador 80286). Está ideado para arquitecturas de 16 bits, y los conectores que se insertan en él son de 98 contactos. El sub-bus de direcciones es de 24 bits (puede direccionar hasta 16 MB) e inicialmente podía transferir información a velocidades de hasta 2 MB/s. Es compatible con el bus ISA inicial (es decir, admite tarjetas de 8 bits, además de las de 16 bits).

MCA *(Micro-Channel Architecture)*
Fue introducido por IBM en 1987 en sus equipos PS/2. Aumenta la velocidad de los bus ISA hasta 20 MB/s, y es para arquitecturas de 32 bits. A diferencia de los buses citados anteriormente y el EISA, es un bus síncrono de forma que las transferencias de información se hacen de forma sincronizada con el reloj del sistema. La especificación inicial se proyectó para velocidades de transferencia de 160 MB/s.

EISA *(Extended Industry Standard Architecture)*
Es un bus ideado por nueve fabricantes de computadores, para arquitecturas de 32 bits (direcciones y datos). La velocidad de transferencia que se puede conseguir es de 33 MB/s. Admite tarjetas de tamaño doble y de consumo de energía eléctrica también doble que las ISA AT y las MCA, y por tanto las ampliaciones son más baratas, ya que la densidad de componentes en la tarjeta puede ser menor, y el consumo de los chips mayor. A diferencia del bus MCA, es compatible con las tarjetas ISA; es decir, pueden conectarse directamente en las ranuras de un bus EISA tarjetas ISA. Este bus sólo puede ser controlado por los microprocesadores 80386, 80486 o Pentium, y es autoconfigurable.

Buses locales
Las tres definiciones más conocidas son: **VLB** (ideada por VESA, un consorcio de más de 120 empresas), **PCI** (*Peripheral Component Interconected*, promovido por Intel), y **Quick-Ring** (propuesta por Apple). Los equipos con VLB y PCI, disponen de tan sólo tres o seis ranuras, respectivamente, ubicadas en las proximidades del procesador, en la propia placa base, para introducir los controladores correspondientes.

Bus AGP *(Accelerated Graphics Port)*
Se introdujo en 1997 por Intel junto con el Pentium II. Está especializado en hardware de gráficos, y trata de aumentar el ancho de banda entre procesador-memoria y el sistema de video, consiguiéndose acelerar la generación de imágenes 3D, la reproducción en pantalla de video en tiempo real, y liberar en parte al bus PCI del trabajo con gráficos.

Futurebus+ *(IEEE 896.1 e IEEE 896.2)*
Es una normalización proyectada para equipos de muy altas prestaciones, que puede considerarse como una evolución de las normas Multibús II y VME. Detalla las características de un panel posterior, para arquitecturas de 64 bits o superiores. Permite la construcción de sistemas multiprocesador (de hasta 32 procesadores) compartiendo memoria.

ATA *(Advanced Technology Attachment)* o **IDE** *(Integrated Drive Electronics)*
Es un bus o interfaz específica para discos duros. Permite que el controlador de disco forme parte de la propia unidad de disco, siendo así digital la comunicación entre el bus y la unidad de disco (y no analógica, como ocurría con anterioridad). La unidad de disco se conecta directamente a la placa base por medio de una versión del mismo bus en forma de cinta o banda flexible que contiene en su interior los hilos conductores. En la versión inicial cada controlador admitía la conexión encadenada de dos unidades, cada una de ellas de hasta 528 MB.

ATA-2 o **EIDE** *(Enhaced Integrated Drive Electronics)*
Es una versión mejorada de IDE, que admite hasta cuatro dispositivos de almacenamiento intercambiables o no (discos duros, CD-ROM y cintas). Cada disco puede llegar a tener una capacidad de 8,4 GB y una velocidad de transferencia de hasta 16,6 MB/s. Se han desarrollado muchas versiones posteriores (ATA 2/3, ATA 4, ATA 5, ATA 6, etc). ATA 6 (también denominado Ultra ATA-100 o Ultra DMA-100) logra velocidades de transferencia máxima de 100 MB/s, y dispone cables de 80 hilos y conectores de 40 contactos.

SCSI (*Small Computer System Interface,* **interfaz de sistema para pequeñas computadas**)
Es un estándar universal para la conexión paralela de una gran variedad de tipos de periféricos. Suele utilizarse para unidades de discos magnéticos y ópticos, disquetes, impresoras y escáneres. El bus SCSI-1 admite hasta 7 dispositivos y fue ideado para entornos UNIX y Macintosh, permitiendo velocidades de transferencia de 5 MB/s, siendo el bus de datos de 8 bits (*narrow SCSI*). La versión SCSII-2 es para datos de 16 bits (*wide SCSI*), con velocidades de hasta 20 MB/s, y admite la conexión de hasta 15 dispositivos. La normalización SCSI-3 admite velocidades de transferencia del doble, 40 MB/s, y permite la conexión de hasta 127 dispositivos. Posteriormente se han desarrollado numerosas versiones, como la Ultra640 SCSI con velocidades de transferencia de 640 MB/s.

USB *(Universal Serial Bus)*
Es un sistema de interconexión serie de muy bajo costo, ideado para gran cantidad de dispositivos periféricos de muy diversas características, y muy cómodo de utilizar con funcionamiento enchufar-y-funcionar (*plug-and-play*). Los cables y conectores de conexión incluyen tan sólo cuatro hilos: dos de transmisión de datos y dos de alimentación, no siendo necesario por lo tanto que el periférico tenga alimentación eléctrica independiente. La versión USB 1.0 admite dos velocidades: baja-velocidad (1,5 Mbits/s) y velocidad-completa (12 Mbits/s) y con él se pueden conectar encadenados hasta 127 dispositivos lentos, tales como ratón, cámaras de fotos, teclado, impresoras, escáneres de baja velocidad y unidades de disquetes. USB 2.0 admite alta velocidad (480 Mbits/s).

FireWire *(IEEE 1394),* o **bus serie de altas prestaciones** *(High Performance Serial Bus)*
Es un bus serie, alternativa a SCSI, al que se pueden conectar en el mismo puerto, en cadena, hasta 63 dispositivos y con el que se logran velocidades de transferencia máximas de 96 Mbits/s (versión

S100), 192 Mb/s (S200) y 384 Mb/s (S400). Puede integrar equipos multimedia tanto de audio como de vídeo. Una de sus líneas es de suministro de energía eléctrica (60 vatios) eliminando la necesidad de utilización del cable de conexión a la red en dispositivos de bajo consumo. Debido a su bajo costo y sencillez, se utiliza, además de en computadores de uso general en sistemas embebidos tales como televisiones de altas prestaciones, cámaras digitales, etc. La versión 1394b admite tres velocidades 800 Mb/s (S800), 1,6 Gb/s (S1600) y 3,2 Gb/s (S3200).

En la Figura 12.6 se muestra un resumen de los distintos tipos de buses descritos, y su ámbito de aplicación.

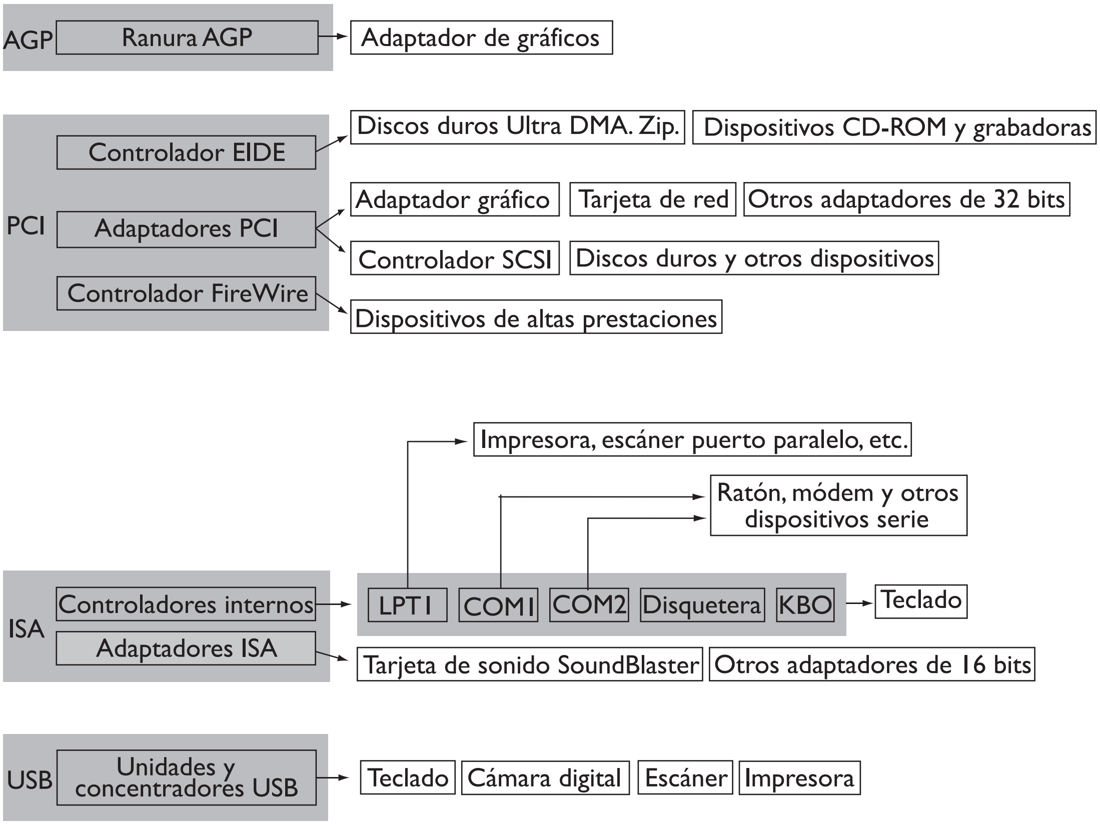


Figura 12.6. *Buses normalizados y ámbitos de utilización.*

12.6 Ejemplo de computador: un PC compatible

A continuación, y con objeto de perfilar mejor los conceptos vistos en las secciones anteriores, vamos a examinar brevemente la estructura interna de un PC compatible.

La mayor parte de los elementos de un computador personal (salvo los dispositivos de entrada/salida) se ensamblan dentro de un chasis. El sistema principal de interconexión es la **tarjeta madre** o **placa base,** que es una tarjeta de circuito impreso (PCB) donde están trazados los buses principales y donde se insertan los dispositivos electrónicos principales del computador y las regletas para conexión de tarjetas de expansión (Figura 12.7).

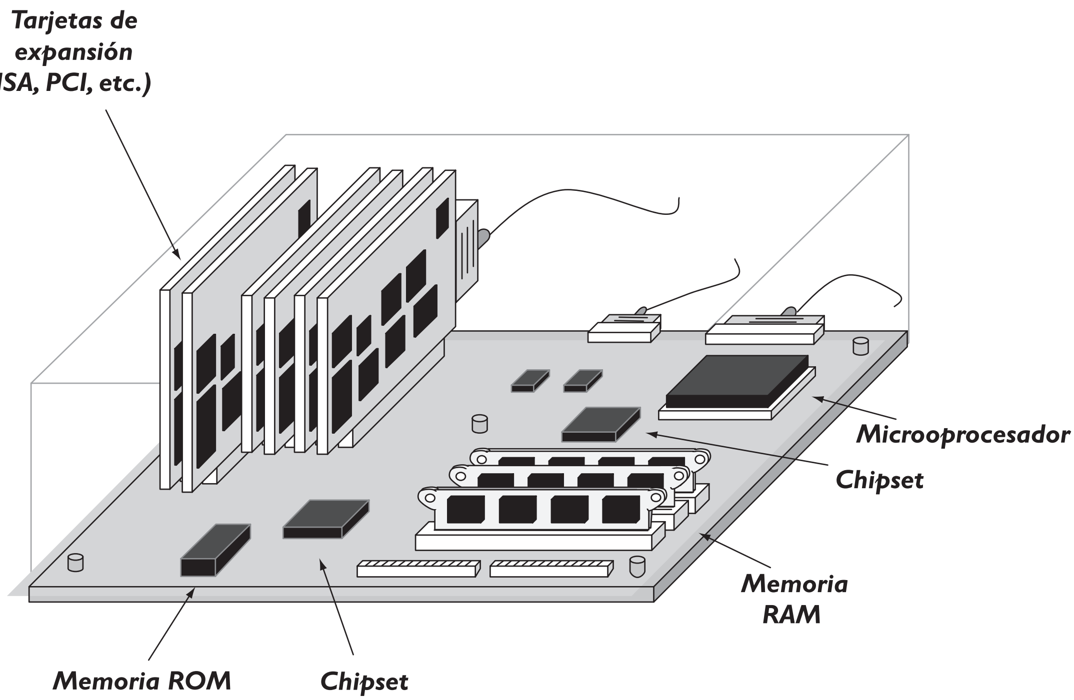


Figura 12.7. *Vista general de la tarjeta base y los elementos insertados en ella, de un PC compatible.*

En los PC actuales los componentes de control principales en la tarjeta base son el microprocesador y dos circuitos denominados genéricamente *chipset*. Los **chipset** son unos circuitos integrados (1, 2 ó 3) que agrupan funciones diversas de la placa base. Dependiendo del tipo de microprocesador contienen el **reloj de tiempo real** y diversos adaptadores (puente PCI, por ejemplo) y controladores (de memoria, DMA, de teclado, del ratón, del acelerador de gráficos, de la interfaz USB, etc.). Estos circuitos determinan la velocidad máxima del bus de memoria y de otros circuitos externos al microprocesador.

En los PC se utilizan distintas tecnologías de memorias RAM que, según vimos en la Sección 10.2.2, se diferencian fundamentalmente en la velocidad y capacidad, las más corrientes son: FPM, EDO, BEDO, SDRAM, RDRAM y Direct DRAM. Comercialmente las memorias se suministran en pequeñas tarjetas de circuito impreso o **módulos de memoria** que agrupan varios circuitos integrados y que se insertan directamente en las regletas de conectores apropiadas de la tarjeta base (Sección 10.2.2).

En la Figura 12.8 se muestra un esquema de un posible montaje[1] de un PC compatible, donde se podría incluir:

- **Procesador**: Pentium 4 570J; 3,8 GHz, con frecuencia del bus frontal de 800 MHz de frecuencia. Dos niveles de caché interna, L1 de microoperaciones (12 Kmicrooperaciones), L1 de 16 KB de datos y L2 de 1 MB.

[1] Para más detalles puede verse: AOpen i865 Family Motherboards, DOC. No.: AX4SGUL-OL-E0309A. http://www.aopen.com/global/.

- **Chipset**: Intel 865. Contiene y sirve de puente de conexión entre procesador, memoria RAM-DDR, puerto AGP y controlador chipset puente de E/S.
- **Chipset puente de E/S:** Chip ICH5, contiene integrados los siguientes elementos: controlador Ultra ATA 100, dos controladores serie ATA, admite 8 puertos externos USB 2.0, control de la interfaz flash BIOS, Control del interfaz PCI, control de CODEC (compresor/decompresor de señales de audio y vídeo), etc.
- **Memoria SDRAM**: módulos DDR (Double-Data-Rate) de 64 ó 128 ó 256 ó 512 MB ó 1 GB. Memoria máxima 2 GB. Permite capturar datos de 128 bits a una frecuencia de 400 MHz.
- **Acelerador de gráficos** (AGP 8x): conector para tarjeta aceleradora de cambios de imágenes en pantalla de altas prestaciones, bus de 32 bits, frecuencia 66 MHz, y por ciclo se transmiten 8 palabras. Velocidad de transferencia: 2.112 MB/s.
- **Conectores PCI:** Cinco ranuras PCI de 32 bits, 33 MHz (132 MB/s), para insertar elementos tales como: tarjeta de vídeo, tarjeta de red, tarjeta de captura de vídeo (TV), y disco duro (SCSII).
- **Ultra DMA 33/66/100 Bus IDE**: Dos conectores que admiten 4 dispositivos IDE en dos canales.
- **Serial ATA (SATA)**: El chipset IC5 dispone de una salida (150 MB/s) para dos puertos con funcionamiento independiente con DMA.
- **ATA-IDE**: El chipset IC5 dispone de una salida ATA 33/66/100 con dos canales que admiten DMA y cada uno cuatro dispositivos IDE (con controlador interior) tales como disco duro (HD), unidad de CD-ROM, unidad grabadora de CD-ROM, y unidad de DVD-ROM.
- **CODEC**, que permite la inclusión en la placa base de un sistema de control completo de grabación y audición de audio.
- **USB 2.0**: Cuatro puertos en el panel trasero y dos interfaces USB 2.0 en la tarjeta, posibilitando la instalación de hasta ocho interfases USB 2.0 para conectar dispositivos tales como, teclado, ratón, módem, escáner, etc.
- **Disquetera:** conector para unidad de disquetes.
- **Conectores del panel frontal:**
  - Conexión para altavoz.
  - Interruptor para reiniciar el arranque del PC (*Reset*).
  - Indicador luminoso de encendido (*POWER LED*).
  - Indicador luminoso de disco duro en funcionamiento (*HD LED*).
  - Entrada para infrarrojos.
  - Indicador de que el sistema está con alimentación eléctrica pero en reposo; es decir, en estado suspendido (*Sleep*).
  - Interruptor para encendido (*Power On*).

Además de los elementos anteriores, dentro del chasis se incluye una serie de ventiladores para que la temperatura de los circuitos integrados no supere un determinado valor, y una fuente de alimentación que transforma la corriente alterna de suministro de energía eléctrica en corriente continua a los niveles de tensión que requieren los distintos circuitos y elementos internos del chasis; éstos en un Pentium 4 son: +5 V (20 A), –5 V (0,5 A), 12 V (8 A), –12 V (0,5 A) , y 3,3 V (14 A); en donde entre paréntesis se indican los valores máximos de la corriente suministrada por cada una de las fuentes. La alimentación del PC requiere una tensión alterna dentro de los límites de 200-240 V y consume del orden de 2,5 A.

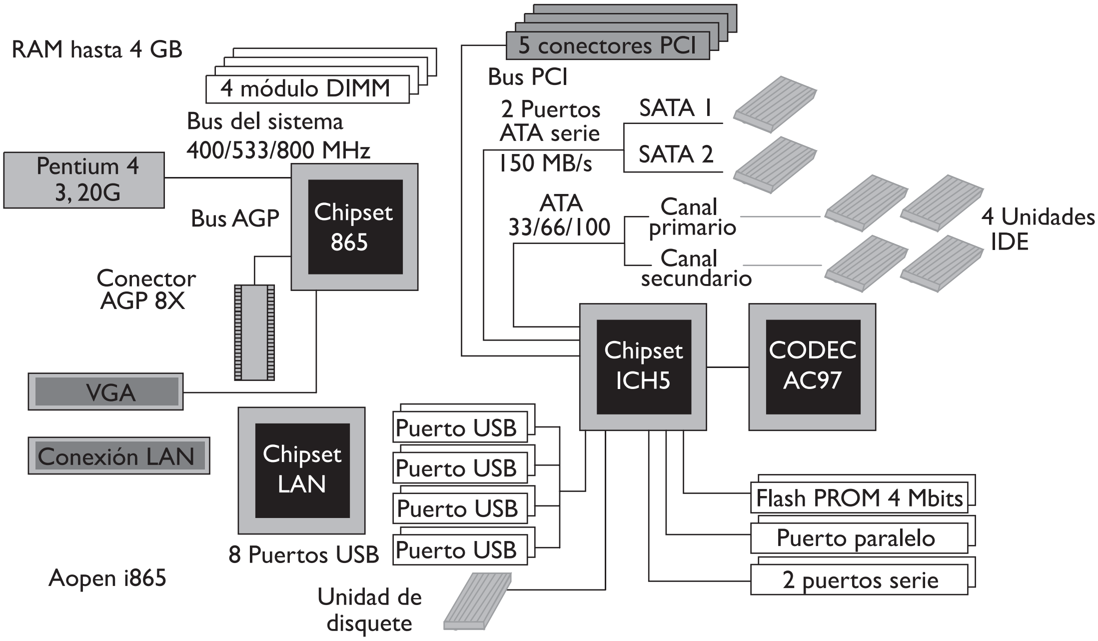


Figura 12.8. *Esquema de un PC compatible, basado en una placa base de la familia AOpen i865.*

En la Figura 12.9 se muestra un esquema de cómo se ubican los distintos elementos en la tarjeta base.

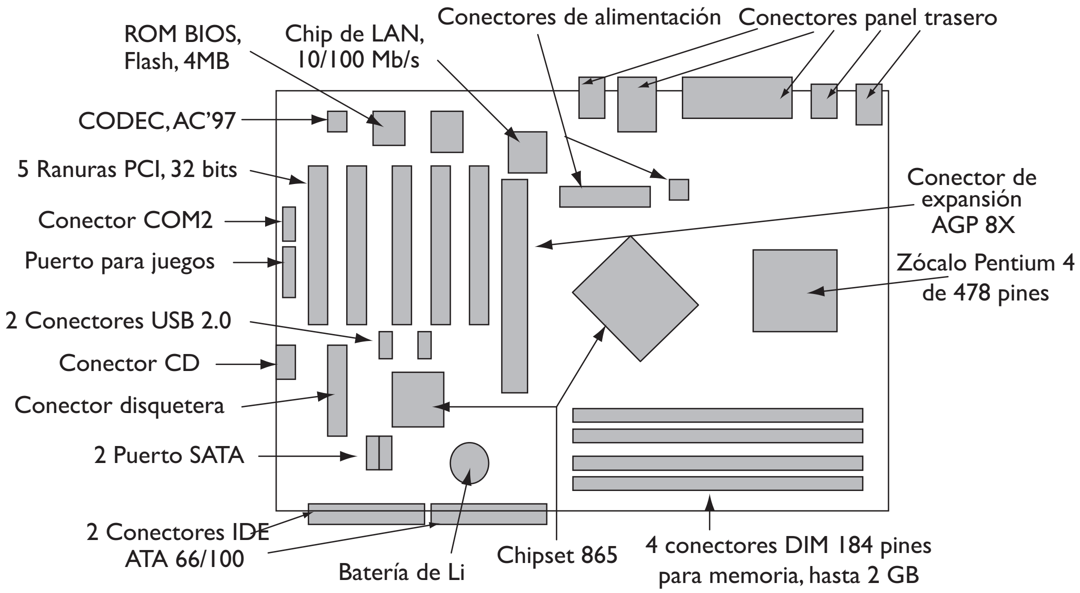


Figura 12.9. *Esquema general de la distribución de los conectores de una tarjeta base de un PC compatible(AOpen i865).*

12.7 Paralelismo en computadores

En esta sección se incluyen conceptos sobre arquitectura de computadores que han mejorado notablemente las prestaciones de los mismos y que se fundamentan en la utilización de paralelismo. Los aumentos de rapidez y potencia en los computadores se han y están logrando no sólo gracias a las mejoras tecnológicas (aumento de la frecuencia de reloj y de miniaturización en los circuitos integrados) sino además por la introducción de nuevos conceptos y diseños relacionados con la *arquitectura* del computador. Las mejoras de prestaciones debidas a la arquitectura se han conseguido fundamentalmente duplicando o segmentando adecuadamente los elementos o unidades del procesador de forma que se puedan ejecutar varias instrucciones a la vez, conociéndose el conjunto de estas técnicas como **procesamiento paralelo**. El paralelismo se ha introducido en muy diversas formas, como se indica en la Figura 12.10 y a continuación se describen.

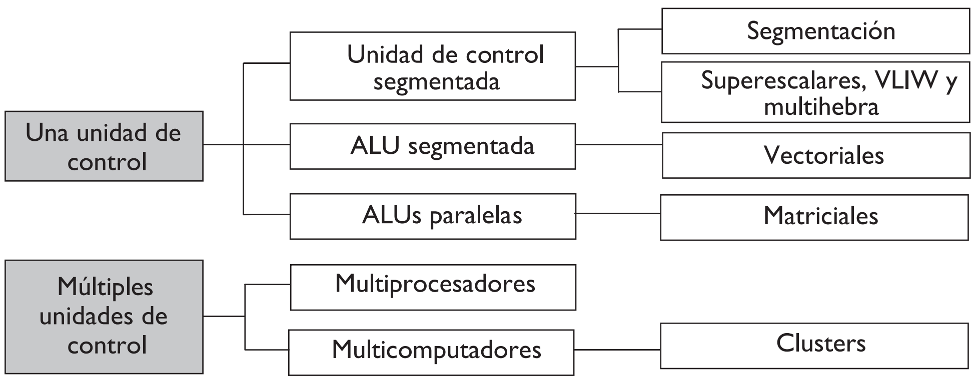


Figura 12.10. *Diversas formas de paralelismo.*

Paralelismo a nivel de instrucciones

Una primera forma de obtener paralelismo es entre instrucciones (ILP, *Instruction Level Parallelism*). Las técnicas que se aplican tratan de reducir el número de ciclos por instrucción que por término medio dedica el procesador a ejecutar una instrucción (CPI), con lo que, según la expresión [12.1], se reduce el tiempo de ejecución de los programas.

Como hemos visto en capítulos anteriores la ejecución de una instrucción individual se realiza en varias etapas, y cada una de ellas consume un ciclo de reloj, de forma tal que es habitual que las instrucciones más sencillas consuman del orden de 3 a 6 ciclos. Estas etapas pueden ser, por ejemplo (Figura 12.11a): captación de instrucción (F), decodificación de instrucción y captación de operandos (D), ejecución de la operación especificada por la instrucción (E), y almacenamiento del resultado (W). No necesariamente todas las instrucciones hacen uso de estas cuatro etapas.

Una forma de acelerar el funcionamiento del procesador es diseñar la unidad de control de forma modular con diversos segmentos o módulos (cuatro en el ejemplo), de forma que cada uno de ellos esté especializado en la ejecución de cada una de las etapas en que se descompone la ejecución de cada instrucción (Figura 12.11a). Con este sistema mientras un segmento se encarga de implementar la etapa que le corresponda de una instrucción, los otros pueden dedicarse a realizar otras etapas de otras instrucciones. En realidad las instrucciones se van ejecutando como en una cadena de producción o montaje, y en un instante dado estarán en

ejecución tantas instrucciones como segmentos contenga la unidad de control. El conjunto de segmentos por los que tiene que pasar una instrucción para su ejecución completa se denomina **cauce.** A lo largo del tiempo, por término medio se podrá ejecutar una instrucción en un ciclo de reloj. Esta técnica se denomina **segmentación de cauce** (*pipelining*) o, abreviadamente, **segmentación**. Puede observarse que, en un funcionamiento normal, puede lograse la ejecución de una instrucción por ciclo ($N_{CI} = 1$).

En el procesamiento con segmentación de cauce, debido a que se inicia la ejecución de instrucciones sucesivas sin haberse completado las primeras de ellas, se presentan dos problemas:

1. Las **dependencias de datos**, consistentes en que cuando una instrucción *B* en su fase *F* necesite un resultado previo generado por una instrucción *A* anterior que no haya llegado a su fase *W* y por tanto éste no esté aún disponible.
2. Las **dependencias de instrucciones**, que se producen en las bifurcaciones o saltos condicionales: si entra en el cauce una instrucción de salto condicional no se conoce las instrucciones que deben seguir entrando en el cauce sin haberse establecido el valor de la condición y sin haberse ejecutado la propia instrucción de salto.

Existen técnicas y métodos para prever y abordar los problemas anteriores que, por supuesto, complican el diseño de la unidad de control.

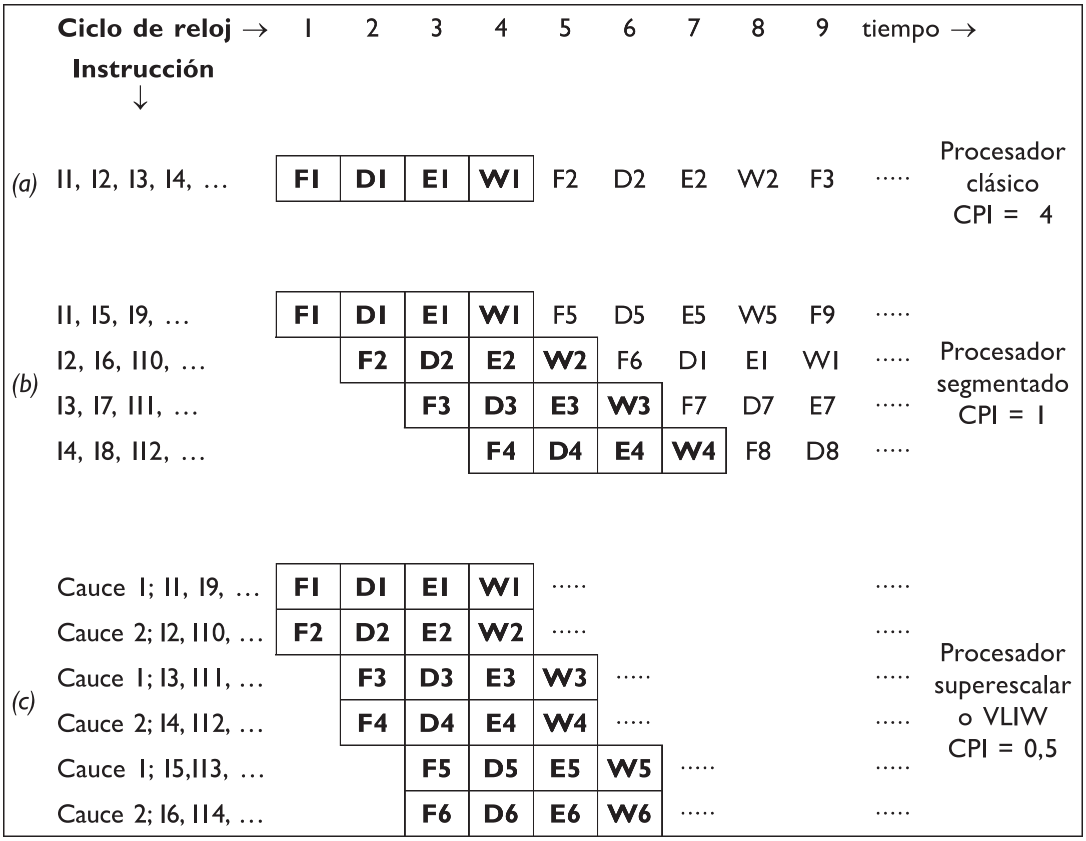


Figura 12.11. *Esquema que muestra la descomposición de una instrucción en distintas fases y su ejecución en un procesador: (a) clásico, (b) segmentado y (c) superescalar o VLIW.*

Se puede lograr también paralelismo dentro del procesador con **arquitecturas superescalares.** Este tipo de computadores se caracteriza por tener varios cauces cuyas etapas pueden trabajar en un instante dado con distintas instrucciones de forma que se ejecutan en paralelo varias instrucciones sucesivas. En el ejemplo de la Figura 12.11c, sin tener en cuenta las dependencias, se podría lograr que por término medio el número de ciclos por instrucción fuese $N_{CI} = 0{,}5$. En definitiva, los procesadores superescalares aumentan la complejidad del procesador al incluir más unidades y al tener que incluir recursos para ir seleccionando del flujo del programa las instrucciones que se pueden ejecutar en paralelo (*descubrir* el paralelismo).

EJEMPLO 12.5

El Pentium tiene dos cauces con cinco segmentos cada uno; mientras que en el Pentium II hay un único cauce pero múltiples segmentos funcionales (varias ALU) que son los que propiamente ejecutan la instrucción. Ejemplos de los primeros procesadores superescalares que se comercializaron son: Power1 de IBM (1990), Alpha 21064 de DEC (1992), PowerPC 601 (1993), Pentium (1993).

Otro enfoque es el de los **procesadores VLIW** (*Very Long Instruction Word*, palabra de instrucción muy larga) que, como los procesadores superescalares, son segmentados que emiten (concluyen) varias instrucciones en cada ciclo y disponen de varias unidades de ejecución donde se pueden ejecutar operaciones simultáneamente. La diferencia fundamental reside en que el compilador crea las instrucciones VLIW incluyendo en cada una de ellas varias instrucciones escalares (las que ejecutan los procesadores superescalares) que el procesador VLIW puede ejecutar en paralelo. Al pasar la responsabilidad de planificar las instrucciones que se pueden ejecutar en paralelo al compilador, los procesadores VLIW no tienen que incluir el hardware que en un procesador superescalar se encargaba de dicha planificación, con lo que se reduce el consumo de potencia y queda liberado espacio en el chip del procesador para incluir otras unidades funcionales pudiéndose incrementar aún más el paralelismo. Un ejemplo de este tipo de procesadores es el Itanium 2 (IA-64).

Como veremos en la Sección 13.3.1, un programa, durante su ejecución, se puede descomponer en unidades denominadas **hebras** ejecutables concurrentemente unas con otras. Así, por ejemplo, un procesador de textos, en el momento de ejecutarse se descompone en pequeños procesos o hebras: una que atiende al teclado, otra que actualiza la imagen en pantalla, otra que atiende al corrector ortográfico, etc. Las hebras se ejecutan independientemente unas de otras. Una mejora adicional en las prestaciones de los procesadores superescalares se obtiene con los **procesadores multihebra**, cuya característica fundamental reside en que cada cauce se encarga de ejecutar una hebra distinta, evitándose así la dependencia entre instrucciones.

EJEMPLO 12.6

El Pentium 4 dispone de hipersegmentación de 20 etapas. Las versiones HT son multihebra (Intel denomina a esta tecnología Hyper-Threading, HT). El procesador admite la ejecución simultánea de dos hebras que pueden corresponder a una misma tarea (programada en varias hebras) o a tareas diferentes, evitándose así las dependencias.

Paralelismo a nivel de procesadores

Para obtener mejoras más substanciales en las prestaciones se han desarrollado diferentes tipos de computadores que contienen múltiples procesadores, clasificados por Flynn según se indica en la Figura 12.12, y que a continuación se describen.

Computadores de un único flujo de instrucciones y un único flujos de datos, SISD *(Single Instruction Stream Single Data Stream)*

Este tipo de sistemas se corresponde con los computadores clásicos, tal y como se han descrito en la Sección 1.2 y se caracterizan por disponer de una única unidad de control (**monoprocesadores**), con segmentación de cauce o no. La unidad de control capta las instrucciones de la memoria, las decodifica y genera las señales de control que implementan la operación correspondiente a cada instrucción que debe realizar la unidad de tratamiento de datos. El flujo de datos se establece a partir de los operandos necesarios para realizar la operación codificada en cada instrucción, que se captan de los registros o de la memoria, y de los resultados generados por las instrucciones que se almacenan en los registros o en la memoria.

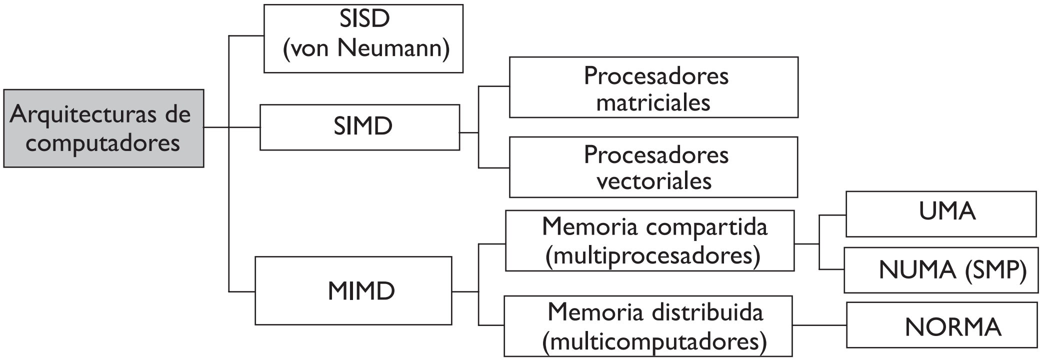


Figura 12.12. *Clasificación de computadores atendiendo al paralelismo a nivel de procesadores.*

Computadores de un único flujo de instrucciones y múltiples flujos de datos, SIMD *(Single Instruction Stream Multiple Data Stream)*

Son computadores con una única unidad de control que procesa las instrucciones de una en una, pero cada una de ellas opera con múltiples datos, de forma que pueden realizar varias operaciones similares simultáneas con operandos distintos. Cada una de las secuencias de operandos y resultados utilizados por las distintas unidades del proceso define un flujo de datos diferente. Hay dos variantes (Figuras 12.10 y 12.12): los procesadores matriciales, que disponen de varias unidades de tratamiento de datos (ALU) que operan simultáneamente con la misma instrucción, y los procesadores vectoriales que realizan operaciones con datos tanto escalares (valores individuales) como vectoriales (datos constituidos por varios elementos) y en los que se dispone de unidades funcionales segmentadas que operan con vectores de n elementos (del orden de 64 a 256, cada uno de 32 ó 64 bits).

Computadores de múltiples flujos de instrucciones y múltiples flujos de datos, MIMD *(Multiple Instruction Stream Multiple Data Sream)*

Disponen de varias unidades de control que decodifican las instrucciones correspondientes a distintos programas. Cada uno de esos programas procesa conjuntos de datos diferentes, que definen distintos flujos de datos. Hay dos tipos de computadores MIMD (Figuras 7.9 y 7.11):

EJEMPLO 12.7

Un ejemplo de procesador vectorial es el NEC SX/6, comercializado en 2002, algunas de sus características son:

- *Frecuencia: 500 MHz.*
- *Número de registros vectoriales: 72.*
- *Elementos por registro: 256.*
- *Unidades de acceso a memoria (carga/almacenamiento): 8.*
- *Unidades funcionales: 8 conjuntos de 5 unidades de procesamiento (suma/desplazamiento, multiplicación, división, operaciones lógicas, enmascaramiento), y unidad para carga/almacenamiento.*

- **Multiprocesadores**. Son sistemas con diversos procesadores pero que comparten la memoria. Los **multiprocesadores con acceso uniforme a memoria** o **UMA** *(Uniform Memory Access)* o **multiprocesadores simétricos (SMP)** se caracterizan porque cada procesador (y cada palabra) tiene el mismo tiempo de acceso a cualquier módulo de memoria, en contraste con los multiprocesadores **NUMA** *(NonUniform Memory Access)* en los que algunos accesos a memoria son más rápidos que otros, dependiendo del procesador que realice el acceso y la palabra a la que se acceda.
- **Multicomputadores**. Son sistemas con múltiples procesadores, cada uno con su memoria local, y no compartiendo memoria, por lo que se les suele denominar **NORMA** (*NO-Remote Memory Access*). La comunicación entre procesadores se realiza a través de una red, enviando mensajes explícitos y esperando sus respuestas. Suelen dividirse en dos categorías: **Procesadores masivamente paralelos, MPP** *(Massively Parallel Processors)* que contienen más de cien procesadores interconectados por una red específica de muy alta velocidad (Cray T3E, IBM SP/2, Intel/Sandia, etc.), y **agrupaciones de estaciones de trabajo** o **de PC**, entre las que se encuentran los **clusters de computadores**, que están formados por PC o estaciones de trabajo convencionales interconectadas por redes de comunicaciones. Los clusters de computadores permiten:
  - Aprovechar las mejoras tecnológicas con mayor rapidez para configurar plataformas paralelas de altas prestaciones.
  - Utilizar hardware estándar para que esas plataformas sean económicas.
  - Hacer uso de tecnología accesible para amplios sectores de la economía.

EJEMPLO 12.8

Ejemplos de multicomputadores son los sistemas: IBM-SP2, y HP AlphaServer SC (clúster de SMP).

Ejemplos de multiprocesadores con acceso no uniforme a memoria (NUMA) son los sistemas: Cray T3E, Cray X1, Origin 3000, HP 9000 Superdome y KSR-1.

EJEMPLO 12.9

Hay sistemas multiprocesador integrados en un único circuito integrado, denominándose **multiprocesadores en un chip** *(***CPM***). Un ejemplo es el Power 4 de IBM, que incluye dos procesadores superescalares a 1 GHz (o más), y cuya velocidad de transferencia con la caché es de 100 GB/s y la del bus de panel frontal 55 GB/s. Hay versiones que incluyen cuatro Power 4 en un módulo multichip MCM con un ancho de banda de 40 GB/s a memoria o a otros módulos.*

Un estudio detallado del paralelismo en computadores puede encontrarse en la referencia [Ort05].

12.8 Conclusión

En el presente capítulo hemos ampliado diversos conceptos sobre la estructura y arquitectura de computadores. De esta forma nos hemos referido a los procesadores RISC y CISC (Sección 12.1) y a distintos tipos de procesadores integrados (Sección 12.2). Los sistemas embebidos han adquirido una importancia fundamental dentro de las aplicaciones de la informática, por lo que a ellos les dedicamos un apartado (Sección 12.3).

Mención aparte merece el análisis de la forma de interconectar las distintas unidades que configuran un computador. Para ello hemos mostrado las estructuras básicas de interconexión (Sección 12.4), y a continuación hemos descrito las características de los buses normalizados más comunes (Sección 12.5), y también hemos estudiado como ejemplo la estructura de un PC compatible (Sección 12.6). Por último (Sección 12.7) hemos introducido el concepto de paralelismo en computadores, tanto a nivel de instrucción como de procesador.

EJERCICIOS

12.1 Completar el diseño esbozado en el Ejemplo 12.3, asignando nombres de variables a todas las entradas y salidas y completando todas las acciones de control. tener presente que siempre cuando el comprensor esté parado y se requiera una acción de él debe de permanecer en el estado de arranque al menos 1 minuto.

12.2 Suponiendo que los puertos de entrada IP02 e IP03 y de salida OP2 y OP3 del CODE-2 se pueden utilizar para el control del climatizador descrito en el Ejemplo 12.3, hacer un programa que implemente las acciones de control obtenidas en el ejercicio anterior (Ejercicio 12.1).

12.3 Considere una versión de CODE-2 que funciona a una frecuencia de 1 GHz. Suponiendo que las transferencias en los buses no ralentizan el funcionamiento de CODE-2 (es decir, en un ciclo de reloj se hace una transferencia de información), hacer una estimación de la velocidad mínima de transferencia (MB/s) en los buses externos de dirección y de datos.

12.4 Suponiendo que el puerto de salida para datos a visualizar en pantalla es el OPF1, hacer una estimación del tiempo que se tardaría en visualizar una imagen que se encuentra en la memoria principal a partir de la dirección A000 y que ocupase 20 KB. (*Sugerencia:* utilizar los resultados del problema anterior, y realizar un programa para realizar la transferencia y calcular el tiempo que tardan en ejecutarse las instrucciones correspondientes).

CAPÍTULO 13

Sistemas operativos

En este capítulo se define en primer lugar el término software y se estudia la estructura del software de un computador, clasificando los distintos programas que se suelen utilizar en un computador típico (Sección 13.1). Después se analizan, con cierta profundidad, los objetivos y el funcionamiento de los sistemas operativos. Para ello se define el concepto de sistema operativo (Sección 13.2), se considera éste como *máquina operativa,* dentro de los niveles conceptuales de análisis de un computador que vimos en la Sección 1.6 (Sección 13.3), y se describe brevemente la evolución de los sistemas operativos, desde la primera generación de computadores a la actualidad (Sección 13.4).

El objetivo fundamental de los sistemas operativos es gestionar y administrar eficientemente los recursos hardware, permitiendo que se ejecuten concurrentemente varios programas, sin que haya conflictos en el acceso de cada uno de ellos a cada uno de los recursos que necesite, y sin que ningún programa monopolice alguno de ellos. La forma en que realiza esta función el sistema operativo se analiza en detalle en las Secciones 13.5 (gestión del procesador), 13.6 (gestión de la memoria principal), y 13.7 (gestión de periféricos). En estas secciones se analizarán conceptos tales como *multiprogramación*, *tiempo compartido*, *memoria virtual*, y *dispositivos de entrada/salida virtuales*, que utilizan los sistemas operativos multiusuario. Finalmente se estudian las funciones de los sistemas operativos en cuanto a la gestión de grandes volúmenes de información, lo que se hace utilizando las abstracciones de archivo y directorio (Sección 13.8).

13.1 Estructura del software de un computador

El **software o soporte lógico de un computador** *es el conjunto de programas asociados a dicho computador.* Dentro de estos programas se incluyen los suministrados por el constructor, los adquiridos en empresas especializadas en venta de programas, y los redactados por los propios usuarios del computador.

Según se muestra en la Tabla 13.1, los programas o componentes que forman el software de un computador pueden agruparse en tres grandes apartados:

Software de control o sistema de explotación
Es el software que controla el funcionamiento de los programas que se ejecutan, y administra los recursos hardware, facilitando el uso del computador de la forma más eficiente posible. Dentro de este apartado se incluye el sistema operativo, el intérprete del lenguaje de control, y el software de diagnóstico y mantenimiento.

Tabla 13.1. *Agrupación de los componentes del software de un computador.*

Estructura del software de un computador		
Software de control	– sistema operativo	• núcleo • planificadores (a corto, a medio y a largo plazo) • módulos de gestión de memoria • módulos de gestión de periféricos • sistema de archivos
	– intérprete del lenguaje de control o concha	
	– programas de diagnóstico y mantenimiento	
Utilidades	– traductores (ensambladores, macroensambladores, compiladores e intérpretes) – cargadores/montadores de programas – editores de textos – rastreadores/depuradores – respaldo de seguridad – compactación de discos – recuperación de archivos – antivirus – compresión de datos – administrador de bibliotecas de programas, etc.	
Software de aplicaciones	– bibliotecas de programas matemáticos y estadísticos – procesadores de textos – programas de presentaciones – programas de hojas electrónicas – sistemas de administración de archivos y de bases de datos – programas para gestión de comunicaciones (correo electrónico, etc.) – visualizadores y navegadores de web – programas de aplicaciones gráficas – programas CAD/CAM – videojuegos, etc – PROGRAMAS DE LOS USUARIOS	

El **software de diagnóstico y mantenimiento** está formado por programas a utilizar por las personas responsables del mantenimiento e instalación del hardware y del software del computador. Con estos programas se pretende localizar automáticamente las averías de

un determinado dispositivo o componente, o las causas de un mal funcionamiento de algún módulo del sistema operativo.

Por ejemplo, un programa de detección de averías de la memoria principal puede escribir el valor $v = 0$ en la posición de memoria $d = 0$, $v = 1$ en la posición $d = 1$, y así sucesivamente. Posteriormente se leen de forma consecutiva las distintas posiciones de memoria y se comprueba si "posición" coincide con "contenido" ($v = d$). Si no es así, evidentemente funcionan mal los circuitos de memoria donde se encuentra la posición d. El programa contiene una tabla que asocia las posiciones de memoria *(d)* con el módulo de memoria *(m)* y el circuito integrado *(c)* donde físicamente se ubican dichas direcciones. El programa de diagnóstico, consultando esta tabla, da a los técnicos de mantenimiento la posición exacta *(m, c)* de los circuitos deteriorados. Los lazos del programa de lectura/escritura/comprobación se ejecutan iterativamente muchas veces; de esta forma podemos obtener las frecuencias con las que se producen los fallos, y así detectar averías intermitentes.

Los programas de mantenimiento sirven para instalar un nuevo sistema operativo, o una nueva versión de él, o para modificarlo por haber realizado cambios en el hardware. Estos programas suelen presentar ventanas de menús o de diálogo, a través de las cuales se especifican los parámetros deseados para el correcto funcionamiento del sistema operativo, y las características del hardware concreto de que dispone el computador.

Utilidades

Son un conjunto de programas de servicio que, en cierta medida, pueden considerarse una ampliación del sistema operativo. Incluye programas para realizar tareas tales como **compactación de discos** o reubicación de archivos dentro de un disco para poder acceder a ellos más rápidamente, compactando los fragmentos libres del disco, **compresión de datos**, o reducción del tamaño de un archivo utilizando algoritmos de compresión (véase Sección 3.8), **gestión de comunicaciones** a través de tarjeta de red o de módem, **Visualizadores y navegadores de internet**, programas para **respaldo de seguridad**, o copia del contenido de un disco fijo en cinta magnética, disquete o CD, **recuperación de archivos** erróneamente borrados, **antivirus, salvapantallas** que evitan imágenes fijas durante largos periodos de tiempo que pueden deteriorar la pantalla, etc. También se incluyen aquí herramientas generales que facilitan la construcción de las aplicaciones de los usuarios, sea cual sea la naturaleza de éstas, tales como **intérpretes, compiladores, editores de texto, y cargadores/montadores.**

Software de aplicación

Incluye programas relacionados con aplicaciones específicas, como pueden ser procesadores de texto, bibliotecas de programas para resolver problemas estadísticos (BMDP, STAT-PACK, SPSS, por ejemplo) o de cálculo numérico (IMSL, MATH-PACK, por ejemplo), sistemas de administración de archivos y de bases de datos, etc. También aquí se incluyen los propios programas realizados por los usuarios. Entre los programas de aplicación es común incluir, entre otros, los que se describen a continuación:

- **Procesadores de textos,** como el Microsoft Word y Corel WordPerfect. Existen editores de texto más perfeccionados que se denominan ***programas de edición electrónica***, que permiten confeccionar y componer periódicos, revistas, etc. Ejemplos de estos programas, que son de gran utilidad en las empresas de artes gráficas y editoriales, son Adobe FrameMarker, Aldus PageMaker y QuarkXPres.
- **Programas de presentación,** como el Microsoft Powerpoint o el Macromedia Flash.
- **Hojas electrónicas u hojas de cálculo,** como el Calc integrada en el OpenOffice, Microsoft Excel integrada en el Microsoft Office y el Gnumeric integrada en el Gnomo Office.

- **Sistemas de gestión de archivos y de bases de datos,** como Oracle, Microsoft Access, dBase, PostgreSQL, MySQL, SQLite, Informix, Microsoft SQL Server, y Fox Pro.
- **Gestores de correo electrónico,** como Pegassus Mail, Eudora, Mozilla Thunberbird y Outlook Express.
- **Programas para gráficas y dibujos.** Se incluyen paquetes o programas de (entre paréntesis de incluyen ejemplos de paquetes comerciales):
  - Rutinas gráficas (por ejemplo, GKS).
  - Programas de dibujo (Visio, Windows Paint, Microsoft Word).
  - Programas de Diseño con Ayuda de Computador (CAD) (Autodesk's AutoCAD, AutoCAD LT, Drafix, CADKEY).
  - Pintura (PaintShop Pro, Adobe SuperPaint, Fractal Design Painter).
  - Manipulación de fotografías (Adobe Photoshop, Corel Photo Paint, Micrografx Picture Publisher).
  - Modelado tridimensional (3-D Studio MAX, Electric Image, SoftImage).
  - Animación y efectos especiales (3-D Studio MAX, LightWave 3D, Electric Image, SoftImage, Kinetix'Character Studio, Alias/wavefront's Power Animator).

  Para más detalles puede verse el Capítulo 9 de la referencia [Nor00].
- **Visualizadores y navegadores de internet.** Son programas que sirven para visualizar y acceder a páginas web. Ejemplos de estos programas son Microsoft Internet Explorer, Netscape Navigator y Mosaic NCSA.
- **Aplicaciones para comunicaciones.** Estas aplicaciones suelen incluir protocolos de comunicaciones así como programas emuladores de terminales, y permite al computador conectarse en una red como terminal de otra, transferir archivos, etc. (Capítulo 18).

Existe una gran cantidad de paquetes comerciales que incluyen de forma integrada la mayoría de las aplicaciones citadas, que se suelen denominar **paquetes integrados de ofimática** o **suites ofimáticas**, aunque el ámbito de uso de los mismas supera con creces el campo de la gestión administrativa. Ejemplos de paquetes integrados son: Microsoft Office, Corel PerfectOffice y Lotus Smart Suite, OpenOffice, Gnome Office, StarOffice, Papyrus OFFICE y ThinkFree Office.

La denominación **paquete** o **software integrado** hace referencia a lo siguiente:

a) Las distintas aplicaciones o utilidades se diseñan con un objetivo y filosofía de utilización común. Existe una *cooperación* entre los distintos módulos para conseguir un fin compartido.

b) Los archivos creados por cualquiera de las aplicaciones pueden ser utilizados por las demás. Es decir, existe una compatibilidad de información de una aplicación a otra.

c) Dentro de una aplicación se puede llamar a otra, de forma que unas pueden auxiliar a las demás. Así, por ejemplo, un software integrado de oficina electrónica permite operaciones tales como:

   1) Representar directamente en un histograma la información de una columna de una hoja electrónica.
   2) Utilizar el procesador de texto para componer un mensaje para el correo electrónico, o para editar un campo de una hoja electrónica.
   3) Combinar un texto generado con el procesador de textos con un archivo creado por el sistema de gestión de archivos para generar directamente cartas personalizadas.

Conviene hacer notar que la clasificación anterior es difusa, en el sentido de que a veces es difícil incluir un programa dado en una de las tres categorías definidas. Además, poco a poco se tiende a incrementar las funciones del sistema operativo.

13.2 Definición de sistema operativo

Un sistema operativo es un *programa* (o conjunto de programas) *de control* que tiene por objeto *facilitar* el uso del computador y conseguir que éste se utilice *eficientemente*.

Es un programa de control, ya que se encarga de gestionar y asignar los recursos hardware que requieren los programas. Los recursos hardware son: el procesador, la memoria principal, los discos, y otros periféricos. Obviamente si varios usuarios están utilizando el mismo computador, el sistema operativo se encarga de asignar los recursos, evitar los conflictos que puedan surgir cuando dos programas requieran los mismos elementos (la misma unidad de disco, o la misma impresora, por ejemplo), y de la seguridad (que un usuario no pueda acceder sin autorización a la información de otro, por ejemplo).

El sistema operativo facilita el uso del computador, haciendo *transparente* al usuario las características hardware concretas de los dispositivos.

El *sistema operativo* también *hace que el computador se utilice eficientemente*. Así por ejemplo, un sistema operativo de multiprogramación hace que cuando un *Programa A* se esté ejecutando, y tenga una operación de E/S, el tiempo muerto de uso del procesador sea aprovechado para la ejecución de otro *Programa B*.

Entre otras, el sistema operativo efectúa las siguientes funciones:

- Facilita el uso del computador y, en general, la comunicación computador/usuario.
- Gestiona y asigna recursos hardware (procesador, memoria y periféricos) a los distintos programas o tareas.
- Gestiona y mantiene los archivos en dispositivos de memoria masiva.
- Apoya a otros programas.
- Protege los datos y los programas, cuestión especialmente compleja en sistemas multiusuario.
- Identifica y autentifica a los usuarios que hacen uso del computador.
- Contabiliza la utilización de los recursos realizada por los distintos usuarios.

13.3 El nivel de máquina operativa

En la Sección 1.6 se indicó que para diseñar o analizar un computador, dada su gran complejidad, se suele considerar según distintos niveles de complejidad. El sistema operativo junto con el hardware del computador definen un nivel de **máquina virtual,** que también se puede denominar **máquina operativa** (Figura 13.1), ya que este conjunto permite utilizar el computador sin tener que conocer muchos de los detalles del hardware y ofrece, como se verá a lo largo del resto de este capítulo, servicios adicionales tales como **memoria virtual, dispositivos de E/S virtuales, multiprogramación,** etc.

El sistema operativo puede considerarse como un programa constituido por una serie de módulos. Estos módulos se lanzan a ejecución por medio de **llamadas al sistema.** Las

llamadas al sistema pueden ser realizadas por los usuarios (directamente, con las órdenes del **lenguaje de control**) o por los programas. En la Tabla 13.2 se incluyen algunas de las llamadas al sistema del sistema operativo MS-DOS.

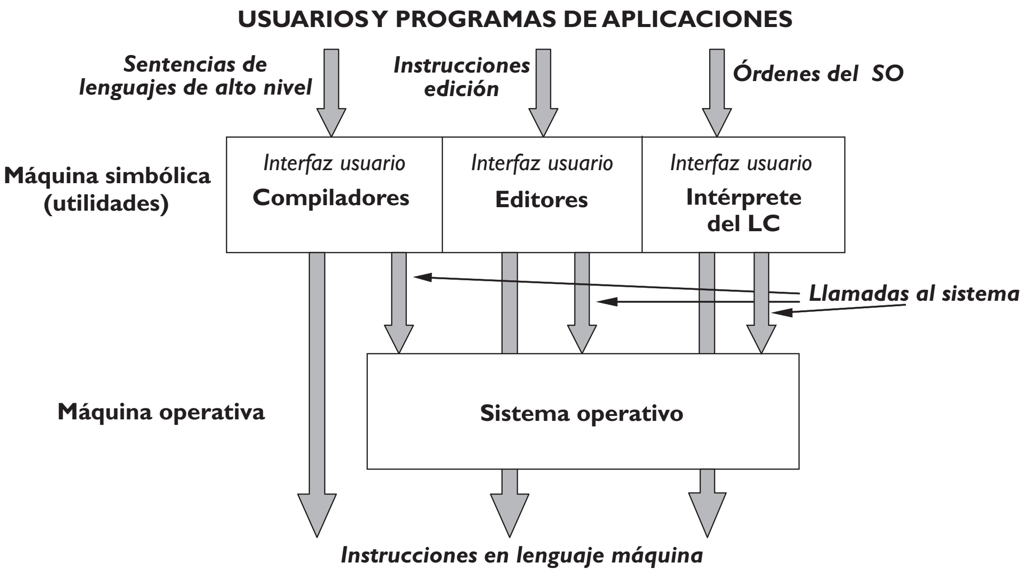


Figura 13.1. *El sistema operativo dentro del contexto de los niveles conceptuales.*

Tabla 13.2. *Algunas llamadas al sistema del MS-DOS.*

load_and_exec	.H'4B	Crear un proceso hijo y ejecutarlo
end_prog	.H'4C	Cancelar un proceso y proporcionar estado
keep_prog	.H'27	Cancelar un programa y dejarlo residente
get_child_status	.H'4D	Obtener el código de retorno del último hijo
alloc_memory	.H'48	Cargar un bloque de memoria
free_allocated_memory	.H'49	Liberar un bloque de memoria
set_mem_blk_sz	.H'4A	Modificar tamaño de un bloque de memoria
ioctl		Lectura y escritura de parámetros del dispositivo
create	.H'3C	Crear un fichero
open	.H'3D	Abrir un fichero
close	.H'3E	Cerrar un fichero
read	.H'3F	Leer datos de fichero a memoria intermedia
write	.H'40	Escribir datos memoria intermedia a fichero
seek	.H'42	Desplazar el puntero del fichero a lugar indic.

Las llamadas al sistema también se conocen con el nombre de **instrucciones virtuales,** para diferenciarlas de las instrucciones máquina del procesador que son las auténticas instrucciones.

Para comunicarse cómodamente con el computador, el sistema operativo va acompañado de módulos que definen la **interfaz del usuario** (Figura 13.1). En general, cualquier programa,

sea cual sea su naturaleza, dispone de una interfaz con el usuario más o menos sofisticada, y la máquina operativa dispone de su propia interfaz, que se fundamenta en la utilización de un **lenguaje de órdenes** (en los antiguos sistemas operativos de colas se denominaba **lenguaje de control de trabajos**). Cuando en un computador introducimos una orden (por ejemplo, dar formato a un disco), ésta es captada por el **intérprete de órdenes** (*shell*), que es un programa independiente del sistema operativo. El intérprete se encarga de traducir o descomponer la orden en llamadas al sistema, de forma que se realicen las operaciones asociadas a la ejecución de la orden.

El lenguaje de órdenes puede tener distintas formas de interacción con el usuario o, en otras palabras, su interfaz con el usuario puede ser de distintos tipos:

- **Interfaz clásica** o **interfaz de línea de órdenes:** las órdenes tienen la forma de líneas de texto que escribe el usuario, utilizando una lexicografía y obedeciendo a unas reglas sintácticas y semánticas claramente preestablecidas.
- **Interfaz de menús:** el usuario no necesita escribir las órdenes, sino que se limita a seleccionarlas con el ratón o el cursor dentro de un menú de opciones presentado en la pantalla.
- **Interfaz con ventanas:** la imagen en pantalla puede subdividirse en pequeñas pantallas (denominadas ventanas), en cada una de las cuales puede visualizarse la interfaz de usuario de una aplicación distinta; con acciones del ratón, puede "abrirse" o "cerrarse" una ventana, moverla, ampliarla, reducirla, etc.
- **Interfaz con iconos:** como característica adicional a los menús, en las ventanas pueden visualizarse pequeños dibujos o **iconos** que simbolizan acciones (metáforas). El usuario se limita a seleccionar con el ratón el icono u objeto deseado para solicitar una acción concreta (por ejemplo, selecciona una "papelera" para borrar un archivo); hay iconos para representar en la pantalla documentos, programas y dispositivos.

En general, una interfaz con ventanas y/o con iconos se denomina **interfaz gráfica de usuario** (**GUI,** *Graphical User Interface)***.** En la actualidad este tipo de interfaz suele incluir:

- **Menús.**
- **Iconos.**
- **Ventanas.**
- **Botones:** al seleccionarlos con el ratón, se realiza una acción.
- **Cajas de diálogo:** son ventanas especiales en las que aparecen distintas opciones que se seleccionan con distintos tipos de botones, facilitando al usuario la instalación de nuevos periféricos, elegir distintas posibilidades en la ejecución de una determinada tarea, insertar información en forma de texto, etc.
- **Barras** que agrupan de forma alineada iconos de: menús, tareas en ejecución, programas de mayor uso, herramientas dentro de un programa o aplicación, etc.; hay también barras de desplazamiento de la imagen dentro de una ventana, para cuando toda ella no cabe en la superficie asignada a la ventana.

Al encender un computador se lanza a ejecución un programa de **autodiagnóstico de encendido** (*Power On Self test*, **POST**), que identifica la memoria disponible, los discos, el teclado, la tarjeta de vídeo, el ratón y los demás dispositivos que lo configuran. Posteriormente se lanza a ejecución el **cargador inicial** (*bootstrap loader*), que a su vez carga un

programa cargador más sofisticado, cuyo objetivo es buscar el sistema operativo y cargar una parte de él (denominada comúnmente **residente**) en la memoria principal. Tanto el programa de autodiagnóstico de encendido como el cargador inicial suelen estar grabados en la memoria ROM del computador. El sistema operativo, una vez arrancado el computador, presenta en pantalla el indicador de petición de entrada del intérprete de órdenes o una imagen (denominada **escritorio**), a la espera de que el usuario inicie su trabajo.

Obsérvese que la interfaz de usuario (sobre todo si es de tipo GUI) es una forma más que tiene el sistema operativo de facilitar el uso del computador, concretamente la comunicación computador/usuario.

13.4 Evolución de los sistemas operativos

En esta sección vamos a considerar en primer lugar (Sección 13.4.1) las distintas etapas en que se puede agrupar en el tiempo los avances en el campo de los sistemas operativos, y posteriormente (Sección 13.4.2) describiremos brevemente las características de los sistemas operativos más difundidos.

13.4.1 Etapas en el desarrollo de los sistemas operativos

Es habitual considerar cuatro etapas o generaciones, en la evolución de sistemas operativos.

Primera etapa

Desde la construcción de los primeros computadores (1943) hasta aproximadamente el año 1955 no existen sistemas operativos. El usuario que deseaba utilizar un computador debía apuntarse en un cuaderno de reservas de horas. Cuando llegaba su turno, el programador, que hacía también las funciones de operador, introducía su programa, y esperaba a que poco a poco fuesen escribiéndose en una impresora los resultados. A partir del inicio de la década de los 50, el programa y los datos se introducían en el computador a través de tarjeta perforada.

EJEMPLO 13.1

Un computador dispone de una lectora de tarjetas de velocidad de lectura de 600 tpm, y de una impresora de 1200 lpm de velocidad de impresión. Si el procesador ejecuta por término medio un millón de instrucciones por segundo, ¿qué tanto por ciento de tiempo actúa el procesador en un bucle de un programa que contiene 100 instrucciones máquina, la lectura de 1 tarjeta y la impresión de 1 línea? (Este ejemplo está tomado de [Dav77]).

El tiempo de ejecución del lazo será:

Lectura de una tarjeta	*0,1000 s*
Ejecución por la CPU de 100 instrucciones	*0,0001 s*
Imprimir 1 línea	*0,0500 s*
Total ciclo	***0,1501 s***

% tiempo de procesador = (0,0001/0,1501) x 100 = 0,066%

Es decir, es como si se aprovechase ¡1 minuto del procesador por cada 24 horas máquina! Y eso que se utiliza una lectora de tarjetas y una impresora considerablemente rápidas.

Con esta forma de trabajar no se obtiene un aprovechamiento adecuado del computador. Cuando el procesador está trabajando, los dispositivos de E/S están inactivos, y cuando un dispositivo de E/S está actuando, las demás unidades, incluyendo el procesador, no trabajan. Las grandes posibilidades operativas del procesador, en cuanto a velocidad de ejecución, no se aprovechan adecuadamente. Esto se pone de relieve con el siguiente ejemplo.

Como resumen de esta primera etapa (1943 a 1955) podemos decir que la introducción y control de la ejecución de programas se hacía manualmente, uno a uno: no existían sistemas operativos.

Segunda etapa

En la segunda etapa (aproximadamente de 1956 a 1963) en vez de utilizar sólo lectoras de tarjetas e impresoras, se utilizan soportes de información intermedios, como cinta magnética.

Inicialmente en cinta magnética sólo se encontraban los traductores, funciones de biblioteca, etc., necesitándose dar por tarjeta únicamente el programa y datos del usuario.

Posteriormente se dio un paso importante. Se utilizó un computador satélite para gestionar las entradas/salidas, y el computador principal (más potente) únicamente leía/escribía sobre cinta magnética. El computador especializado en entradas salidas se limitaba a ir formando en una cola de trabajos a ejecutar, grabando **imágenes de tarjeta.** Cuando la cinta estaba llena se montaba en el computador central, quien iba leyendo sucesivamente los trabajos de la cola y ejecutándolos uno a uno. Los resultados a imprimir los escribía en otra cinta magnética, obteniéndose así por cada programa un **archivo de impresión** grabado en cinta. Una vez completada una cinta con los resultados de los programas (archivos de impresión), se montaba en el computador satélite que iba imprimiendo sucesivamente los distintos archivos de impresión.

EJEMPLO 13.2

Suponiendo que la velocidad de lectura de una imagen de tarjeta, desde cinta, es de 1000 t/s (es decir, la velocidad de transferencia de salida de la unidad de cinta es aproximadamente 80 KB/s) y que la velocidad de grabación de una línea-registro es 200 l/s, calcular el % de utilización del procesador en el bucle del programa del Ejemplo 13.1, con el mismo procesador.

El tiempo de ejecución del lazo será:

Lectura de una imagen de tarjeta	*0,0010 s*
Ejecución por la CPU de 100 instrucciones	*0,0001 s*
Grabación de la imagen de una tarjeta	*0,0500 s*
Total ciclo	***0,1501 s***

% procesador = (0,0001/0,0031) x 100 = 3,2%

Es decir, se obtiene un aprovechamiento relativo unas 50 veces mayor que en el caso del Ejemplo 13.1.

Para poder operar como anteriormente se ha indicado, el paquete de tarjetas o trabajos del usuario debía contener unas tarjetas adicionales denominadas tarjetas de control, escritas de acuerdo con un **lenguaje de control de trabajos.** Las tarjetas de control, para diferenciarlas

de las del programa y de datos, empezaban por un carácter o caracteres identificadores de su naturaleza de tarjetas de control: $, @, //, etc.

Así había unas tarjetas que identificaban los límites de cada trabajo. Estas tarjetas, por ejemplo, podían ser:

Primera tarjeta: ***$JOB*** (e información adicional indicando el nombre del trabajo, código del usuario, tiempo máximo de ejecución, etc.)

Última tarjeta: ***$END***

El operador organizaba el trabajo incluyendo, por ejemplo, en una misma cinta todos los trabajos con programas en FORTRAN, en otra los trabajos con programas en COBOL, etc.

Cuando se pasaban al IBM-7090 los trabajos en FORTRAN, se montaba en otra unidad la cinta con el compilador y bibliotecas de FORTRAN, cargándose en el computador cuando fuese necesario. El trabajo del usuario a procesar incluía otras tarjetas de control, indicando qué traductor debía utilizarse (***$FTN***, tarjeta para compilar en FORTRAN, ***$ASM***, para ensamblar, etc.), en qué momento debía procederse a ejecutar el programa (tarjeta ***$RUN***), etc.

El procedimiento de funcionamiento descrito, desde el punto de vista del usuario-programador, es mucho más sencillo que el de la primera etapa. Pero es necesario un control automático del flujo o cola de trabajos. En efecto, este tipo de funcionamiento, denominado por **lotes** o **lotes-serie** o **cola-serie** o por **cola de trabajos** (en inglés, *batch* o *batch-serial*), es posible gracias a la existencia de un sistema operativo; es decir, un programa de control. Este programa de control que planifica los trabajos se suele denominar monitor. El **monitor** en un sistema de lotes-serie debe hacer fundamentalmente tres cosas:

a) Identificar (a partir de su carácter o caracteres iniciales) e interpretar las tarjetas de control. Una vez interpretada una tarjeta de control el monitor da paso a una rutina que realiza las funciones asociadas a ella (por ejemplo, carga el compilador de FORTRAN, pasando el control a él para que traduzca las tarjetas que siguen a ***$FOR*** a código máquina).
b) Intervenir al final de la ejecución de un trabajo (es decir, cuando identifica una tarjeta ***$END***) para cerrar el archivo de impresión del trabajo que acaba, e iniciar (pasar el control) al siguiente trabajo de la cola; es decir, hace una secuenciación automática de trabajos.

Poco a poco los monitores de lotes se perfeccionaron, incluyendo rutinas de gestión de E/S y bibliotecas de programas utilizables por cualquier usuario, módulos controladores del tiempo máximo de procesador (para evitar que un programa se esté ejecutando por tiempo indefinido, a causa de un error de programación), módulo de contabilidad del uso del computador por los usuarios, cargadores, etc.

Esta segunda etapa la podríamos denominar de **procesamiento de trabajos en serie** o por **lotes-serie**, o también de utilización de dispositivos de E/S fuera de línea, o de **dispositivos de E/S virtuales**, que se analizarán más adelante (Sección 13.7).

Tercera etapa

La tercera etapa (1963 a 1979) se caracteriza fundamentalmente por el perfeccionamiento del sistema de trabajos por lotes (con la técnica de lotes paralelos o de secuencias múltiples), el desarrollo de sistemas operativos en multiprogramación (1963), que permite a los usuarios

trabajar en forma interactiva/conversacional, la introducción del concepto de memoria virtual (1972), y la utilización compartida de dispositivos de entrada salida (*spooler*). Estas técnicas las describiremos en las Secciones 13.5, 13.6.5 y 13.7.

Cuarta etapa

La cuarta etapa (aproximadamente entre 1980 y la actualidad) se caracteriza por el desarrollo de sistemas operativos para computadores personales (CP/M, MS-DOS, OS/2, etc.) y para estaciones de trabajo (UNIX, Windows-NT, etc.).

También se desarrollan sistemas operativos de red, de **multiprocesamiento simétrico** o **SMP** (*Symetric Multiprocessing*) y sistemas operativos distribuidos. Los **sistemas operativos en red** no se diferencian sustancialmente de los sistemas operativos convencionales, ya que pueden considerase como una ampliación de éstos con programas de control de interfaz con la red, permitiendo establecer una sesión de trabajo con un computador remoto, acceder a los recursos de éste y copiar archivos de un computador a otro [Sil99]. Los **sistemas de multiprocesamiento simétrico** actúan en computadores con múltiples procesadores, todos ellos realizando las mismas funciones y que comparten la misma memoria principal y facilidades de E/S e interconectados por un bus u otro esquema interno de conexión. Un sistema operativo SMP distribuye y planifica los programas y trabajos entre los distintos procesadores, sincronizando las relaciones entre ellos, dando la apariencia al usuario de disponer de un solo procesador. Los **sistemas operativos distribuidos** son mucho más complejos ya que permiten que un usuario ejecute, de forma transparente, como si estuviese actuando con un único computador, uno o varios programas en varios computadores (**clúster de estaciones de trabajo**, por ejemplo) y con datos distribuidos en diferentes servidores de archivos, lo cual requiere que el sistema operativo disponga de un sistema de gestión de archivos distribuido y de sofisticados algoritmos de planificación que paralelicen adecuadamente el trabajo a realizar. Un usuario trabajando en red con un sistema operativo distribuido tiene la ilusión de que está actuando con una única memoria principal, una única unidad de disco y un único sistema de archivos, aunque en realidad todos esos elementos se encuentran en distintos computadores.

Los sistemas operativos modernos tratan de cubrir los objetivos marcados en la Sección 13.2 teniendo en cuenta que en la actualidad la mayoría de los recursos informáticos (procesadores, programas, información, etc.) no se encuentran localmente sino en la red.

13.4.2 Sistemas operativos de mayor difusión

Según se comentó en la Sección 1.3, inicialmente cada fabricante de computadores (IBM, DEC, Data General, Apple, Sperry, etc.) desarrollaba sus propios sistemas operativos, que incluso, dentro de la misma empresa, podrían diferir de una serie de computadores a otra. Esto dificultaba considerablemente el trabajo de los programadores y usuarios, en general, ya que, con gran frecuencia, al cambiar de computador tenían que aprender un nuevo sistema operativo y realizar las adaptaciones oportunas de todas las aplicaciones al nuevo equipo. Este panorama ha cambiado en la actualidad debido a que en el mercado se han impuesto sistemas operativos que no son propietarios de un determinado fabricante de computadores sino que son realizados por terceras personas o empresas especializadas en software de forma que pueden ser utilizados en equipos muy diversos. Los más relevantes son los siguientes:

Sistemas operativos de Microsoft

Son los sistemas operativos dominantes en el mercado de microcomputadores. El primero de ellos es el **MS-DOS** (*Microsoft Disk Operating System*) y debe su difusión a que fue adoptado por IBM al inicio de la década de los 80 como el sistema operativo estándar para el IBM-PC. En la Figura 13.2 se muestra un esquema de la evolución de este sistema operativo. El MS-DOS inicial era un sistema operativo para microprocesadores de 16 bits (de Intel), monousuario y tenía una interfaz de usuario de línea de órdenes. A inicios de los 90 se comercializó el Microsoft *Windows 3.0* que disponía de una GUI con gestión de menús, y era capaz de cargar en memoria más de un programa a la vez. En 1995 Microsoft comercializó el *Windows 95*, que contiene una GUI basada en iconos, y es un sistema operativo de 16/32 bits con multiprogramación apropiativa (puede suspender temporalmente la ejecución de un trabajo para ejecutar otro, véase Sección 13.5) y con memoria virtual. Posteriormente se comercializaron las versiones Windows 98 y Windows ME (Milenium), en 1998 y en 2000, respectivamente. Conviene indicar que las denominaciones *Windows* en realidad hacen referencia a la GUI: para identificar completamente a las nuevas versiones de Microsoft hay que facilitar también la versión del sistema operativo propiamente dicho; por ejemplo, una versión concreta es*: Windows 98 MS-DOS 7.0.*

Paralelamente al desarrollo de las versiones Windows, en 1993 Microsoft comercializó el **Windows NT** (*Windows New Technology*), diseñado fundamentalmente para estaciones de trabajo potentes y servidores de red con procesadores de 32 bits. En 2001 se disponían de tres alternativas:

- **Windows 2000 professional:** está orientado a estaciones de trabajo en red. Puede utilizarse con diversos procesadores (Intel, Alpha y PowerPC). Es un sistema operativo monousuario y multiprogramación. Su interfaz de usuario es muy parecida a Windows 95, pero ofrece mucha más seguridad (identificación y contabilidad de usuarios) y tolerante a fallos (aísla unos programas de otros, y duplica en línea el contenido de los discos de forma que si falla uno, automáticamente actúa el de respaldo).
- **Windows NT Server** (servidores): es una versión ampliada del Windows NT Workstation, multiusuario y proyectada para actuar en servidores de archivos y de impresoras, y otros sistemas, a los que se accede a través de redes de área local (LAN) o internet. Incluye funciones de seguridad para grupos de usuario, autentificación de usuarios y control de acceso a los recursos compartidos de la red. Admite estructuras RAID de discos (Sección 10.3.2).
- **Windows CE (Consumer Electronics):** es una versión simplificada de las otras versiones de Windows NT ideada para equipos informáticos miniaturizados, como computadores móviles (asistentes digitales personales, computadores de bolsillo, organizadores personales, etc.), televisores conectables a internet, etc.

En el año 2001 se comercializó en Windows XP, que supuso la unificación de la línea clásica de los sistemas operativos de Microsoft (Windows 3.1/95/98/98SE/ME) con la de Windows NT (NT 3.1, 4.1 y Windows 2000), véase Figura 13.2. El Windows XP contiene el núcleo del Windows NT y se comercializó en dos versiones: Home y Proffesional. Posteriormente se lanzó al mercado el Windows 2003.

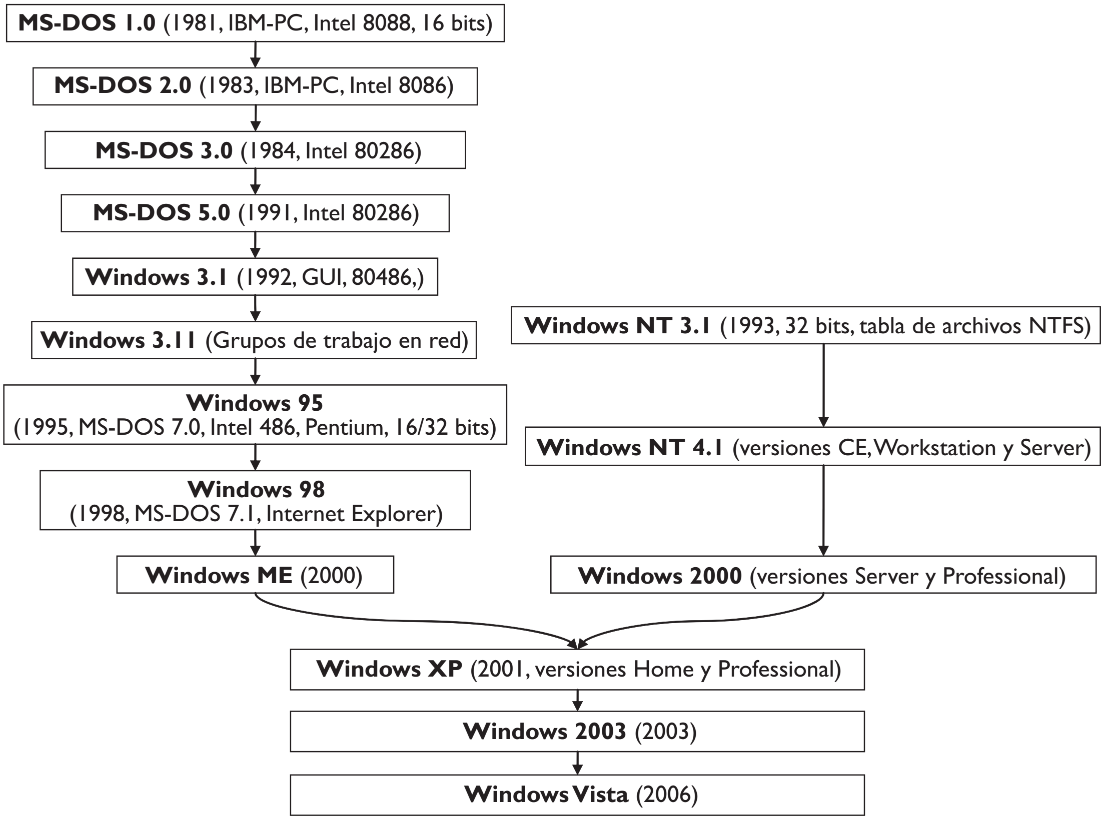


Figura 13.2. *Pasos más relevantes en la evolución de los sistemas operativos Microsoft.*

UNIX

El sistema operativo UNIX inicialmente se desarrolló en los Bell Labs de la AT&T y fue comercializado en sus inicios por Honeywell y, a los comienzos de la década de los 90, por Novell. En la Figura 13.3 se incluye un esquema simplificado de la evolución de este sistema operativo. Una de las versiones más conocidas es la desarrollada en el Campus de Berkeley de la Universidad de California. Algunas de las principales características del UNIX son las siguientes:

- Desde un punto de vista técnico UNIX puede considerarse como un conjunto de familias de sistemas operativos que comparten ciertos criterios de diseño e interoperabilidad. Esta familia incluye más de 100 sistemas operativos desarrollados a lo largo de 35 años. No obstante, es importante señalar que esta definición no implica necesariamente que dichos sistemas operativos compartan código máquina o cualquier propiedad intelectual.
- Puede funcionar en multitud de computadores, desde en los supercomputadores Cray hasta en PC. Debido a ello puede considerarse como un auténtico sistema operativo estándar y la columna vertebral de Internet. Su mayor uso se da en estaciones de trabajo para usuarios CAD y CAM (estaciones de trabajo de Sun Microsystems, Hewlett-Packard, IBM y Silicon Graphics).

- Es un sistema operativo de multiprogramación, multiusuario y multiprocesamiento, pudiéndose utilizar también para procesamiento en tiempo real (estos conceptos se introducirán en la Sección 13.5).
- Resulta más difícil de aprender y de usar que los MS-Windows y Mac.

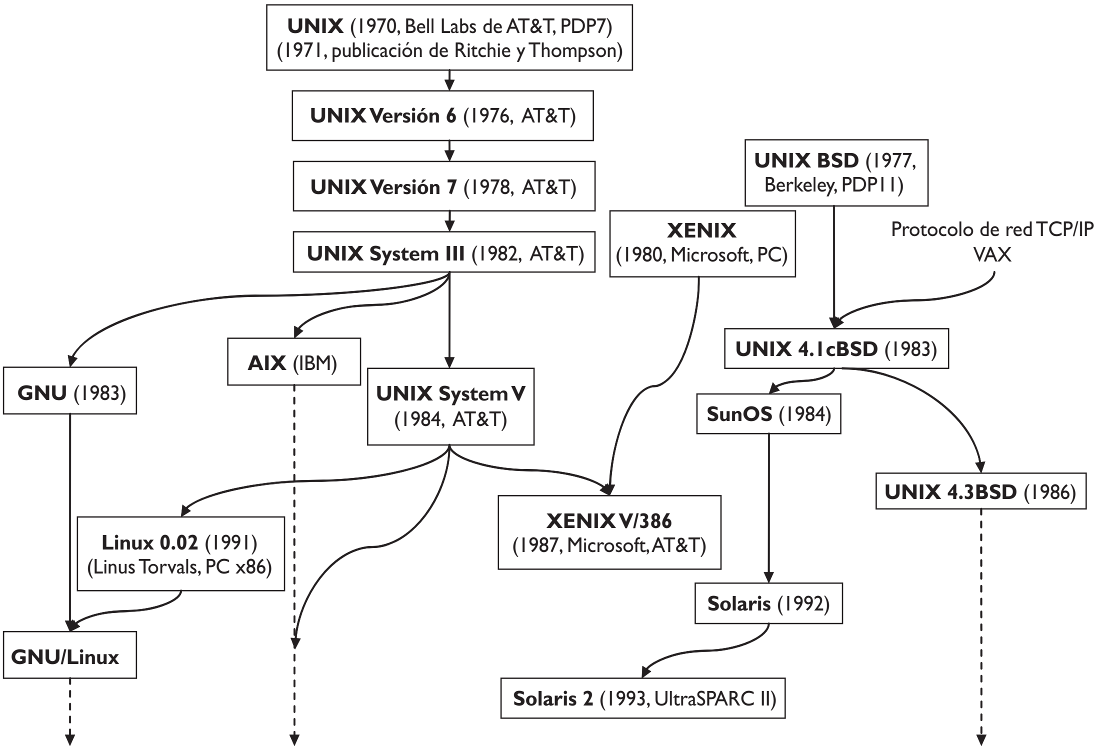


Figura 13.3. *Pasos relevantes en la evolución del sistema operativo UNIX.*

Un estudiante de informática de la Universidad de Helsinki (Finlandia) llamado Linus Torvalls concluyó en 1991 (con 23 años de edad) un sistema operativo que denominó Linux, versión 0.01. Este sistema operativo fue concebido como una versión PC compatible y sustancialmente mejorada del sistema operativo Minix (muy parecido al UNIX), descrito por el profesor Tanenbaum en sus primeros textos de sistemas operativos [Tan98]. Aunque Linux no es un programa de dominio público, se distribuye con una licencia GPL (*General Public Licence*) de GNU, de forma que los autores no han renunciado a sus derechos, pero la obtención de su licencia es gratuita, y cualquiera puede disponer de todos los programas fuente, modificarlos y desarrollar nuevas aplicaciones basadas en Linux, teniendo derecho a regalar éstas e incluso a venderlas [Sil06, Tac00]. La condición de la licencia radica en que nadie puede registrar como propiedad intelectual los productos derivados de Linux y los usuarios que adquieran (por compra o gratuitamente) el código modificado tienen derecho a disponer también del código fuente correspondiente. Linux hoy día puede considerarse como un clónico de UNIX desarrollado por multitud de personas distribuidas por todo el mundo [Tac00].

Otros sistemas operativos

Hay otros muchos sistemas operativos, entre los que se pueden destacar:

- **MacOS**, o sistema operativo de los Macintosh. Desarrollado a mediados de los 80, es sin lugar a dudas el precursor de los sistemas operativos con interfaz gráfica, siendo muy fácil de usar y el preferido por empresas de publicidad y de artes gráficas, y realizadores de aplicaciones multimedia. Su principal inconveniente es que sólo funciona en los PC tipo Macintosh y sus compatibles (microprocesadores Motorola), no existiendo tanta disponibilidad de software como para los IBM-PC y compatibles (microprocesadores Intel).
- **OS/2**, desarrollado a partir de 1982 inicialmente de forma conjunta por IBM e Intel, es un sistema operativo monousuario y multiprogramación ideado para los microprocesadores Intel. Actualmente sólo es comercializado y mantenido por IBM.
- **MVS** (*Múltiple Virtual Storage*), es uno de los sistemas operativos más sofisticados desarrollado por IBM para grandes computadores, es de tipo multiprogramación, multiusuario, funcionando tanto de forma interactiva como por lotes, y con memoria virtual.

13.5 Gestión del procesador

Nos centraremos en esta sección en el estudio de la administración y gestión del procesador, tanto en sistemas de monoprogramación como de multiprogramación. También se describirán distintas variantes de estos modos de procesamiento (Sección 13.5.2), la técnica de intercambio entre memoria principal y disco (Sección 13.5.3) y los distintos estados por los que pasa un proceso desde que se introduce en el computador hasta que finaliza su ejecución (Sección 13.5.4).

Existen dos formas básicas de trabajar con un computador: **por lotes** (o cola de trabajos, Sección 13.4) y en forma **interactiva**. En esta última el procesador está constantemente atendiendo al usuario en una forma que podría denominarse de *diálogo*.

La gestión del procesador por parte del sistema operativo se centra en el concepto de proceso. Un **proceso** *es un programa en el que se ha iniciado su ejecución, y que es tratado por el sistema operativo como un todo*. Un programa por sí sólo es un ente pasivo, mientras que un proceso es un ente activo.

Memoria principal

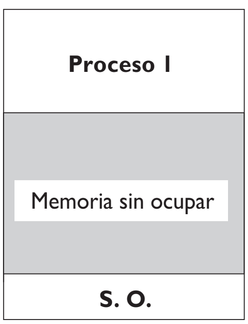


Figura 13.4. *Ejemplo de ocupación de memoria en un sistema de monoprogramación.*

13.5.1 Monoprogramación

Los primeros sistemas operativos se denominaban de **monoprogramación** o **serie**, en ellos se ubicaba un programa en memoria principal, además del sistema operativo (Figura 13.4), y hasta que no finaliza la ejecución de dicho programa no empezaba a ejecutarse otro. En la Figura 13.5 se indican los intervalos de tiempo de procesador y periféricos invertidos por el proceso P1. En los tiempos marcados en la línea *Sistema Operativo*, el procesador está bajo el control de la parte más básica del sistema operativo denominada **núcleo** (o *kernel*), encargada de gestionar los recursos hardware a través de servicios de llamada al sistema. En los intervalos señalados en la línea *Proceso 1*, el procesador está ejecutando el proceso (realizando *cálculos*) y durante el intervalo *AB* se está realizando una operación de E/S, por ejemplo la lectura de un registro grabado en disco, etc.

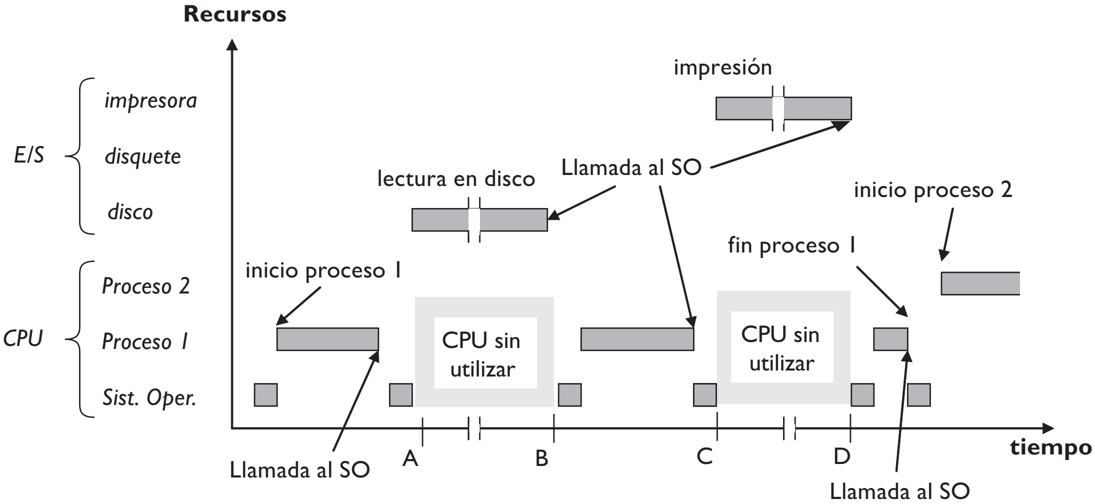


Figura 13.5. *Esquema de distribución del tiempo entre el procesador y periféricos de E/S en la ejecución de dos procesos (P1 y P2) en un sistema monoprogramación.*

Un sistema operativo de monoprogramación realiza las siguientes funciones:

- Siempre que un programa realiza una operación de E/S hace una llamada al sistema operativo, para que éste genere las instrucciones máquina concretas de la E/S.
- Cuando finaliza una operación de E/S, el periférico correspondiente genera una interrupción que provoca una llamada al sistema operativo, y continúa la ejecución del programa.
- Cuando acaba la ejecución de un programa, el computador queda listo para aceptar un nuevo programa, ya sea interactivo o de la cola serie. En este último caso, el **planificador a largo plazo** del sistema operativo (también denominado **planificador de trabajos**), selecciona (de acuerdo con algún criterio de prioridad) un programa de la cola, lo carga en la memoria principal e inicia su ejecución.

Claramente se observa que con este sistema:

- La memoria casi siempre queda sólo parcialmente ocupada (cabrían más programas en ella, Figura 13.4); es decir, *se desaprovecha la memoria principal.*

- Siempre que hay una operación de E/S, el procesador no se utiliza (intervalos *AB* y *CD*, en la Figura 13.4); estos tiempos suelen ser mucho mayores (3 ó 4 órdenes de magnitud) que los tiempos del procesador; en consecuencia *se desaprovecha el procesador*.
- De igual modo, *los periféricos están infrautilizados,* ya que el programa en ejecución en un instante dado obviamente sólo puede utilizar uno de los recursos hardware.

En consecuencia, la monoprogramación no es eficiente.

Para conseguir mayor eficiencia en la utilización de los distintos recursos que ofrece un computador se desarrolló la multiprogramación. La sección siguiente se dedica a describir los conceptos básicos de la misma.

13.5.2 Multiprogramación

Nos centraremos en esta sección en el estudio de la administración y gestión del tiempo del procesador en sistemas de multiprogramación y de tiempo compartido.

13.5.2.1 Conceptos básicos

En el contexto de la multiprogramación, los componentes típicos de un proceso son:

1. Código máquina del programa del usuario a ser ejecutado.
2. Datos del programa: es la parte modificable del espacio del usuario y contiene las variables y parámetros del programa.
3. Pila: cada proceso tiene asociadas una o varias pilas, utilizadas para almacenar direcciones de retorno de subrutinas u otros parámetros.
4. **Bloque de control del proceso** (**PCB**, *Process Control Block*), que contiene datos que el sistema operativo necesita para controlar el proceso (identificador, estado, privilegios y prioridades del proceso, identificador del usuario, contenidos de los registros del procesador, etc.).

El sistema operativo UNIX introdujo el concepto de **hebra** (*thread*), soportado por los sistemas operativos modernos. Un proceso puede descomponerse en distintas hebras o tareas diferentes e independientes, que se pueden ejecutar "en paralelo" *(concurrentemente)* entre ellas e incluso con las de otros procesos. Cada hebra tiene asociado su estado, prioridad, privilegios, etc. Los Windows de Microsoft asocian a cada ventana de carpeta visualizada en la pantalla una hebra independiente, de forma que, por ejemplo, pueden realizarse en paralelo dos copias de archivo en ventanas diferentes. También, por ejemplo, al lanzar a ejecución el procesador de textos Word se generan distintas hebras, cada una de ellas con distinto cometido: una para visualizar el texto en la pantalla, otra para atender las entradas del teclado, otra para el corrector ortográfico, otra para visualizar los menús, etc. La mayoría de conceptos que se exponen en esta sección relacionados con procesos se pueden aplicar a hebras. Por comodidad, en las descripciones que siguen utilizaremos el término *proceso,* pero con más rigor deberíamos decir *hebra.*

Se dice que un proceso (o hebra) está en **estado preparado** cuando el procesador puede iniciar o continuar su ejecución. Se dice que un proceso entra en **estado de bloqueado** cuando el procesador no puede continuar trabajando con él a causa de tener que esperar a la realización de una operación de entrada/salida o a algún otro evento de naturaleza similar. En los intervalos de tiempo *AB* y *CD* de la Figura 13.5 el proceso *P1* se encuentra en estado bloqueado. Un proceso se dice que está en **estado activo** cuando el procesador está ejecutando instrucciones del mismo (*P1* está activo en los intervalos marcados en la línea Proceso 1 de la Figura 13.5).

Un **sistema operativo multiprogramación** carga en memoria principal un programa de la cola serie y todos los programas interactivos que se le presenten (Figura 13.6), y el **planificador de trabajos a corto plazo** (o **distribuidor**) del sistema operativo asigna el procesador sucesivamente a los procesos preparados de forma que el procesador se aproveche al máximo y los distintos procesos vayan avanzando en su ejecución sin necesidad de que finalice completamente uno para iniciar la ejecución de otro.

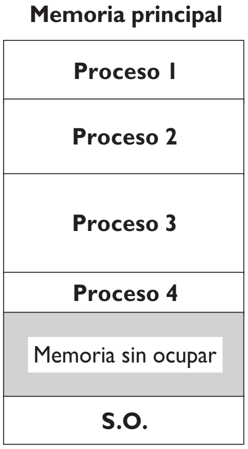


Figura 13.6. *Ejemplo de ocupación de memoria en un sistema de multiprogramación.*

En definitiva, se aprovechan los tiempos muertos del procesador que se tienen con la monoprogramación (*AB* y *CD*, en la Figura 13.5), los tiempos muertos en periféricos y el espacio de memoria principal no ocupado por el proceso o sistema operativo.

Obsérvese que con la multiprogramación la ejecución de los procesos se va solapando en el tiempo y, desde el punto de vista del usuario, como la velocidad interna del procesador es muy grande, parece que esos procesos van ejecutándose simultáneamente (en paralelo). En realidad no es así ya que al existir un único procesador, éste únicamente puede estar trabajando con un solo proceso (el proceso activo). Con rigor se dice que el procesador está ejecutando *concurrentemente* (en vez de *simultáneamente*) varios procesos: en un intervalo de tiempo determinado se han estado ejecutando alternativamente *a trozos* varios procesos ubicados en la memoria central.

Siempre que un proceso se bloquea o concluye, el núcleo del sistema operativo toma el control del procesador para realizar las acciones oportunas. Cuando hay multiprogramación y se detiene la ejecución de un proceso, *Px,* para dar turno a otro, *Py,* (que presumiblemente se había interrumpido previamente), el sistema operativo realiza un **cambio de contexto** consistente en:

1. Actualizar el bloque de control *PCB-Px* del proceso *Px* interrumpido; es decir, se cambia su estado (de activo a bloqueado o preparado) y se salvaguardan los contenidos de los registros del procesador, biestables indicadores, punteros a archivos de discos, y punteros de las pilas, etc.
2. Restaurar los contenidos de los registros del procesador, biestables indicadores, punteros a archivos de discos, punteros de las pilas, etc, con los valores del *PCB-Py* del proceso

Py que va a continuar su ejecución, y cambiar el estado del proceso a activo; de esta forma continua sin problema la ejecución del proceso *Py* previamente interrumpido.

Obsérvese que de esta forma el proceso *Px,* cuando el distribuidor le dé nuevamente el turno, podrá reanudar su ejecución justo en el punto exacto en que se interrumpió.

Para medir la eficiencia de un sistema se suelen utilizar los siguientes parámetros:

- **Rendimiento:** es el número de trabajos ejecutados durante un periodo de tiempo preestablecido (12 horas, por ejemplo). Representa una medida del aprovechamiento del computador, en su totalidad.
- **Tasa de utilización del procesador:** es, para un determinado tiempo, el tiempo de uso del procesador con respecto al tiempo total:

$$P = \frac{tiempo\ de\ uso\ de\ procesador}{tiempo\ total}$$

- **Tiempo de ejecución de un proceso,** e: es el tiempo transcurrido desde que se inicia su ejecución hasta que concluye. Se puede obtener como suma de los siguientes tiempos:
  1) tiempo de procesamiento (suma de tiempos en que el proceso está activo), *s,*
  2) tiempo de E/S (suma de tiempos en que el proceso está bloqueado), w_1, y
  3) tiempo de espera (suma de tiempos en que el proceso está preparado, esperando a que le llegue el turno), w_2.
- **Tiempo en máquina**, *q*: es el tiempo que tarda el sistema en responder a la solicitud de realización de una tarea determinada. En procesos por lotes se suele denominar ***tiempo de retorno,*** y es la suma de:
  1) tiempo de espera en cola de entrada, w_0
  2) tiempo de ejecución, *e*, y
  3) tiempo en la cola de salida (*spooler*), w_3.

  En procesos interactivos se suele denominar ***tiempo de respuesta,*** y se define como el intervalo de tiempo entre el instante en que el usuario da una orden y el instante en que se visualiza la respuesta.
- **Coeficiente de respuesta:** es el conciente

$$R = \frac{tiempo\ en\ máquina}{tiempo\ de\ procesamiento} = \frac{q}{s}$$

y representa una medida de la lentitud del computador en responder al usuario.

13.5.2.2 Asignación del procesador a los procesos

Teniendo en cuenta cómo se toma la decisión de ir alternando la ejecución de los procesos, la multiprogramación se puede realizar de dos formas básicas: no apropiativa y apropiativa.

Multiprogramación no apropiativa *(Nonpreemptive)*
En este caso, una vez que un proceso está activo, continúa ejecutándose hasta que:

1) termina, o
2) se bloquea por el inicio de una operación de E/S o de cualquier otro servicio solicitado al sistema operativo, o
3) el propio proceso hace una llamada al sistema operativo para ceder el procesador a otro proceso.

En este último caso se dice que el sistema es de multiprogramación cooperativa. En el momento de interrumpirse la ejecución del proceso el distribuidor, de acuerdo con algún algoritmo de planificación, da el turno a otro de los procesos pendientes que se encuentren en el estado preparado.

Este modo de cambiar la asignación del procesador se denomina también multiprogramación clásica, por ser el que inicialmente se utilizó. En la Figura 13.7 se representan tres procesos, además de sistema operativo, en ejecución concurrente. Cuando el proceso *P1* se bloquea (por una operación de E/S) el distribuidor pasa el turno de ejecución a otro de los procesos en memoria principal que esté en estado preparado (al proceso *P2*, por ejemplo). Para ello se produce una interrupción que provoca una conmutación de contexto entre los procesos *P1* y *P2*, con el consiguiente consumo de tiempo de procesador por parte del distribuidor, 5 milisegundos, por ejemplo. Cuando el distribuidor vuelva a dar el turno al proceso interrumpido (*P1*), por haber finalizado la operación de E/S (es decir, haber salido del estado de bloqueado y entrado en el de preparado) y haberle llegado su turno, debe producirse otra conmutación de contexto en el sentido contrario al anterior, ya que tienen que restituirse desde la zona auxiliar de memoria (donde se encuentra *PCB-P1*) los contenidos de los registros salvados, quedando el procesador tal y como estaba en el instante que interrumpió este proceso (*P1*), y preparado, por tanto, para proseguir su ejecución. Estas operaciones ocurren sucesivamente para cada uno de los procesos existentes en memoria principal (*P1, P2, P3, P1, P2,* etc.). Compárese la Figura 13.7 con la Figura 13.5 (monoprogramación) y compruebe como el proceso *P1* acaba al mismo tiempo, pero en ese mismo momento ya se ha ejecutado parte de los procesos *P2* y *P3*, debido a que se han aprovechado los tiempos muertos del procesador y de los periféricos que se producen con la monoprogramación.

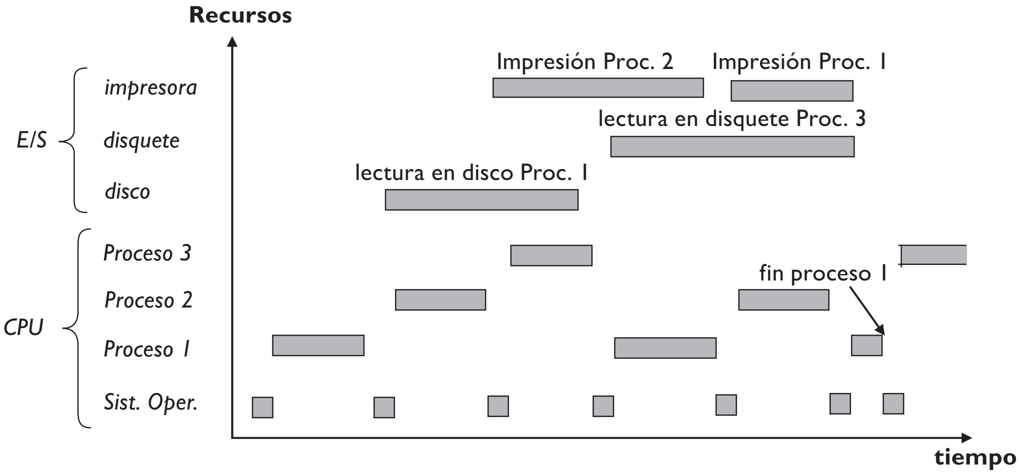


Figura 13.7. *Esquema de distribución del tiempo entre el procesador y periféricos de E/S en la ejecución de tres procesos (P1, P2 y P3) en un sistema multiprogramación no apropiativa.*

Los sistemas operativos MS-DOS (hasta la versión con Windows 3.x) y MacOS utilizan el modo multiprogramación no apropiativa. Supóngase, por ejemplo, que se tiene un proceso (el *spooler*, véase Sección 13.7) de impresión de un archivo de gran longitud denominado

censo. Sin utilizar multiprogramación, mientras durase la impresión, nuestro PC quedaría bloqueado; sin embargo, con la multiprogramación, el proceso de impresión se ejecutaría en el fondo (*background*) y se podría ejecutar concurrentemente con otro proceso, como editar un texto, en primer plano (*foreground*). El sistema operativo da prioridad al proceso en primer plano, y durante los intervalos de tiempo en que se bloquee (larguísimos, por cierto, pues son los comprendidos entre dos pulsaciones sucesivas de teclas) va avanzado la impresión del documento censo. Podríamos enviar varios trabajos a ejecutar en el fondo. En la multiprogramación no apropiativa cooperativa los programas, en puntos determinados pueden hacer una llamada al sistema operativo para ceder el procesador de forma que el distribuidor pueda dar el turno a otro proceso.

EJEMPLO 13.3

Las versiones de Windows 3.x y el MacOS son de multiprogramación no apropiativa cooperativa.

Multiprogramación apropiativa o preferente *(Preemptive)*
La multiprogramación en modo no apropiativo, descrita anteriormente, tiene el inconveniente de que un proceso con mucho tiempo de procesador (mucho *cálculo*) y pocas E/S puede monopolizar el procesador hasta que acabe su ejecución. Los sistemas de multiprogramación apropiativa resuelven este problema ya que con ellos no se espera a que un proceso pase al estado de bloqueado (o a que el propio proceso dé opción al paso de la ejecución de otro) para que el distribuidor lo detenga y dé el turno a otro proceso que esté preparado.

En la **multiprogramación apropiativa** el sistema operativo puede interrumpir en cualquier momento el proceso activo *apropiándose* del procesador con objeto de dar paso a otro proceso que esté preparado.

El sistema operativo detiene la ejecución de un proceso de acuerdo con algún criterio preestablecido, por ejemplo cuando transcurre un determinado tiempo o cuando se presenta un proceso de mayor prioridad, como veremos a continuación. Las decisiones de cuándo se detiene la ejecución de un proceso y de cuál de los procesos preparados pasa a activo se efectúa de acuerdo con un determinado algoritmo de planificación. Este tipo de multiprogramación es usado por Windows 95/98, Windows NT, OS/2, Unix y Linux.

EJEMPLO 13.4

Los sistemas operativos Windows 95/98, Windows NT, OS/2, Unix y Linux utilizan multiprogramación apropiativa.

13.5.2.3 Algoritmos de planificación

Si existen varios procesos preparados, un problema que debe resolver el distribuidor es elegir a cuál de ellos darle el turno (pasar a activo). El **distribuidor** se encarga de implementar un **algoritmo de planificación**.

Uno de los algoritmos de planificación de mayor interés (por su antigüedad, sencillez y amplio uso) es el de **turno rotatorio** (*round robin*). A cada uno de los procesos en memoria se le asigna un intervalo de tiempo fijo, o periodo T, o quantum (usualmente de 100 a 10 milisegundos) (Figura 13.8), y se realiza un cambio de contexto de un proceso activo a otro preparado cuando al activo o se le acaba el quantum o se bloquea.

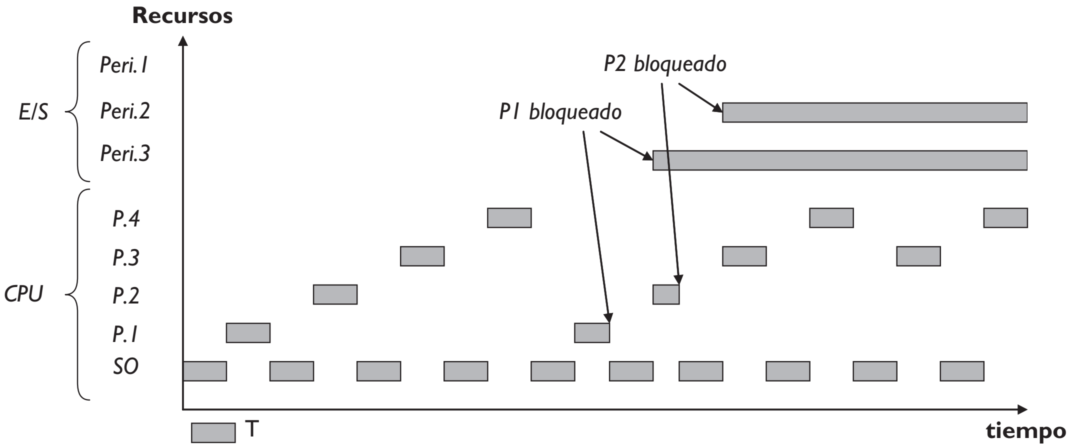


Figura 13.8. *Esquema de distribución del tiempo entre el procesador y periféricos de E/S en la ejecución de cuatro procesos (P1, P2, P3 y P4) en un sistema monoprogramación con turno rotatorio.*

La asignación puede hacerse con ayuda de un temporizador, de forma que cuando transcurre un determinado periodo de tiempo se produce una interrupción que lanza a ejecución el **gestor del reloj de tiempo real**, que entre otros objetivos sirve de referencia para generar el tiempo asociado al quantum.

EJEMPLO 13.5

La elección del periodo del quantum es problemática, ya que si es demasiado pequeño (10 milisegundos, por ejemplo) hay que hacer muchas conmutaciones de contexto (cada 15 milisegundos se pierden 5 ms en cambio de contexto, por ejemplo) y la eficiencia del procesador es baja (66,6%), pero si es grande (100 milisegundos, por ejemplo) los tiempos de respuesta para cada usuario pueden llegar a ser muy bajos: si se estuviesen ejecutando concurrentemente 20 procesos, al pulsar una simple tecla podríamos obtener respuestas tan lentas como de 2 segundos.

Un proceso suele tener asociada una prioridad. Pueden existir **prioridades estáticas**, dadas, por ejemplo, en función de la relevancia de los procesos (o usuarios), o **prioridades dinámicas**, que asigna y cambia el sistema operativo, o **prioridades mixtas**, mezcla de las dos anteriores.

Otro algoritmo de interés es el de **planificación por prioridad,** según el cual el distribuidor da el turno al proceso preparado con mayor prioridad asignada por el propio planificador. Existen varios criterios de asignación de prioridades. A veces, para que el de mayor prioridad

(y por tanto activo) no monopolice el uso del procesador, a cada interrupción del reloj de tiempo real (cada 20 ms) se le baja su prioridad (**planificación con realimentación**). Se conmuta al siguiente proceso cuando alguno de los preparados en cola supere la prioridad del proceso activo.

EJEMPLO 13.6

El algoritmo de planificación por prioridad es usado, entre otros, por los sistemas operativos OS/2, Unix, Linux y Windows NT.

Para obtener un aprovechamiento equilibrado de todos los recursos del computador y obtener mayor productividad es un buen criterio utilizar el algoritmo ***HRRN*** (dar el turno al de mayor coeficiente de respuesta, *Highest Response Ratio Next*) con el que se da prioridad a los procesos que tienen más tiempo de E/S en relación con su tiempo de procesador. Una forma de llevar a la práctica este criterio es dar al proceso P_i la prioridad $1/f_i$, siendo f_i la fracción de tiempo del último quantum asignado a P_i que realmente ha consumido el procesador.

EJEMPLO 13.7

Supóngase que los quanta son de 80 ms, y el proceso P_7 sólo consume de él 20 ms (por haberse bloqueado a causa de una operación de E/S). El planificador le asignará una prioridad de $p_7 = 80/20 = 4$. Si el proceso P_8 hubiese consumido sólo 10 ms de su quantum de 80 ms, obtendría una prioridad de $p_8 = 80/10 = 8$.

Otros algoritmos de planificación son el **FCFS** (primero en llegar, primero en procesar, *First Come First Served*), **SPN** (dar el turno al de menor tiempo de procesador, *Shortest Process Next*), y **SRT** (dar el turno al que le queda menos tiempo de procesador, *Shortest Remaining Time*) cuyas denominaciones son autoexplicativas. Sobre los algoritmos citados es conveniente hacer dos aclaraciones:

1. Algunos de ellos pueden utilizarse en los dos modos de planificación: no apropiativo y apropiativo. En este último caso se utiliza el reloj de tiempo real, y a cada interrupción se toma una decisión de planificación, deteniendo, en su caso, el proceso activo en ese momento.
2. Los algoritmos SPN y SRT presuponen que el planificador conoce de antemano el tiempo de procesador que necesitará cada proceso. Esto no supone ningún problema con el procesamiento por lotes, ya que en estos casos el usuario debe incluir en las órdenes del lenguaje de control una estimación máxima de dicho tiempo. También el planificador puede mantener un archivo histórico de procesos para, a partir de él, hacer una estimación del tiempo previsible para los programas previamente procesados.

13.5.3 Modos de procesamiento

Existen distintos tipos de sistemas operativos con distintos modos de procesamiento, algunos de los cuales se han analizado anteriormente. A continuación, para tener una visión de conjunto, se incluye un resumen de los más relevantes.

- **Multiprogramación.** Se tienen en la memoria principal distintos procesos a los que el sistema operativo va asignando el procesador alternativamente en el tiempo, sin necesidad de que finalice completamente uno para iniciar la ejecución de otro. A veces a estos sistemas se les denomina sistemas multitarea, reservándose la denominación de multiprogramación para sistemas multitarea que dispongan de técnicas apropiadas de protección de memoria y de control de concurrencia para permitir el acceso compartido a dispositivos de E/S y archivos. Si las unidades de procesamiento son hebras, se denomina sistema multihebra.
- **Sistema multiusuario.** Sistema de multiprogramación que prevé el uso concurrente de distintos usuarios, con identificación, autentificación y control de los mismos.
- **Procesamiento serie de cola (***lotes serie***).** Al introducir un trabajo en el computador se incluye en una cola, no iniciándose su ejecución hasta ser seleccionado por el sistema operativo, de acuerdo con las disponibilidades de memoria y un esquema de prioridades. El objetivo del sistema operativo es maximizar el uso del procesador, sin preocuparse de los tiempos de respuesta obtenidos por los usuarios.
- **Procesamiento paralelo de cola** (*lotes paralelo*). El sistema operativo clasifica los trabajos de la cola en diversas clases, de acuerdo con intervalos de prioridad, tiempos de procesador previstos, ocupación de memoria prevista, etc. e introduce en la memoria principal para su ejecución concurrente un trabajo de cada una de las clases de la cola (los de mayor prioridad de cada clase), Figura 13.9.

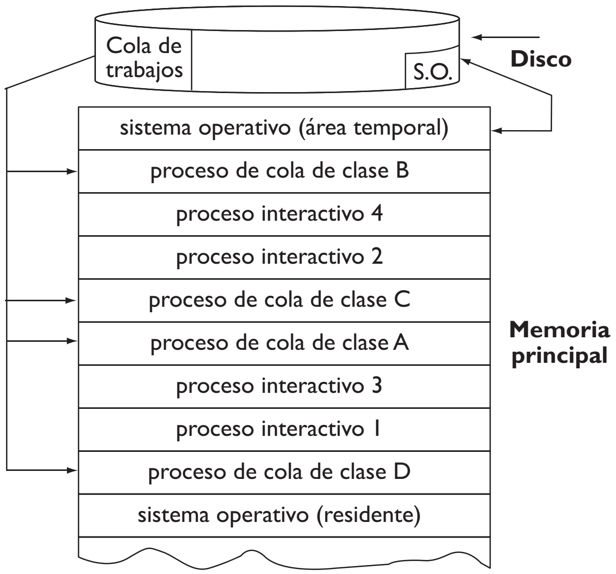


Figura 13.9. *Los sistemas operativos más completos permiten procesar concurrentemente varios trabajos por lotes y procesos interactivos.*

- **Tiempo compartido.** Sistema de multiprogramación y multiusuario que gestiona procesos interactivos, a través de los cuales el usuario interactúa directamente con el computador casi de forma inmediata, dándole la ilusión de que está trabajando él sólo con el computador. El objetivo del sistema operativo es optimizar el tiempo de respuesta obtenido por los usuarios, para lo cual debe utilizar un algoritmo de planificación adecuado (apropiativo con turno rotatorio, por ejemplo).

- **Tiempo real.** El concepto de *tiempo real* hace referencia a que el sistema debe dar imprescindiblemente la respuesta dentro de un límite de tiempo preestablecido. Este tiempo puede ser pequeño o grande, dependiendo de la aplicación. Los sistemas en tiempo real se usan ampliamente en control industrial, sistemas embebidos (en un coche o avión, por ejemplo), redes digitales de transmisión de datos, control de vuelo, aplicaciones militares (guía de misiles, etc.) y simuladores de tiempo real. Los sistemas operativos en tiempo real deben ser capaces de responder a eventos (interrupciones) que se pueden producir asíncronamente y con una gran frecuencia (a veces miles por segundo) en unos plazos de tiempo previamente especificados. Frecuentemente son sistemas multiprogramación en los que los algoritmos de planificación son del tipo no apropiativo con planificación por prioridad, con los que siempre se ejecuta la tarea de mayor prioridad, y no se interrumpe hasta que finalice, a no ser que se genere otra de prioridad mayor. Hay dos modalidades: **tiempo real estricto** (*hard*) y **tiempo real no estricto** (*soft*), dependiendo de si la restricción del tiempo de respuesta es estricta (respuesta para en hinchado de la bolsa de aire de un coche, por ejemplo) o sólo deseable (respuesta a una transacción desde un cajero de un banco), respectivamente.

EJEMPLO 13.8

Los sistemas operativos UNIX (por ejemplo la versión SVR4), Linux (siguiendo la norma POSIX.1b) y Windows NT disponen de clases o niveles de prioridades especiales para procesos de tiempo real no estricto.

- **Procesamiento de transacciones.** Sistema de tiempo compartido (y a veces de tiempo real no estricto) que generalmente implica realizar cambios en una gran base de datos satisfaciendo ciertos requisitos en cuanto a tiempo de respuesta y a recuperación de ciertos tipos de fallos. Las entidades bamcarias utilizan estos sistemas ya que deben asegurar que las transacciones no se pierdan, se procesen rápidamente y se minimice el coste de cada transacción. Otro ejemplo es el sistema AMADEUS para reserva de billetes de avión. Los sistemas de procesamiento por transacciones realizan conjuntos de operaciones como una unidad de trabajo *atómica*; esto quiere decir o que ejecuta completamente todo el conjunto o no se realiza ninguna de las operaciones que la componen, no pudiendo finalizar en un estado incompleto según se pone de manifiesto en el siguiente ejemplo.

EJEMPLO 13.9

Un ejemplo de transacción es la adquisición de una entrada de un concierto a través de la web. El conjunto (atómico) de operaciones a realizar sería:

a) Consultar en la base de datos si hay localidades libres.
b) Asignar al usuario una localidad.
c) Hacer el cargo del precio de la entrada en la tarjeta de crédito del usuario.

El sistema de transacciones debe garantizar que, si por cualquier motivo, se interrumpe el proceso, por ejemplo entre la operación 2 y 3, o no se pueda realizar con éxito la operación 3 (por no tener crédito el usuario) se deshagan las órdenes ya ejecutadas con objeto de dejar la base de datos como si la transacción nunca se hubiese realizado. De esta forma se garantiza la ***integridad de los datos*** *del sistema.*

- **Sistema de multiprocesamiento.** Estos sistemas están ideados para actuar con dos o más procesadores trabajando simultáneamente (Sección 12.7). El sistema operativo distribuye los distintos procesos o hebras entre los diferentes procesadores. Con el multiprocesamiento asimétrico un procesador principal controla el comportamiento global de todos los procesadores, pudiendo utilizarlos como si fuesen dispositivos colocados en su bus. El procesador principal puede sobrecargarse en las tareas de control y distribución de trabajo: al añadir un procesador más al sistema la potencia de procesamiento del global no se incrementa linealmente. Con el multiprocesamiento simétrico, por contra, se consigue un incremento aproximadamente lineal de prestaciones según aumenta el número de procesadores; pero requiere sistemas operativos y compiladores muy sofisticados para poder aprovechar las ventajas del multiprocesamiento. Un sistema de multiprocesamiento multihebra puede asignar las distintas hebras de uno o varios procesos a los distintos procesadores del sistema.

EJEMPLO 13.10

Hay versiones de UNIX para procesamiento asimétrico y simétrico. Windows NT admite procesamiento simétrico. Por otra parte, el Pentium 4, versión HT, es un microprocesador superescalar multihebra en el que en cada uno de sus cauces puede ejecutarse en paralelo una hebra distinta.

13.5.4 Intercambio memoria principal/disco

El número de trabajos ejecutándose concurrentemente depende (entre otros factores) de la capacidad de la memoria principal. En un computador de 32 MB de memoria principal, si el sistema operativo residente en memoria ocupa 4 MB, y cada usuario de terminal interactivo ocupa 2 MB, como máximo podrán estar trabajando n usuarios, con n dado por:

$$32 = 4 + 2 \cdot n$$

es decir, $n = 14$ usuarios. No podrían utilizar el computador concurrentemente más de 14 usuarios.

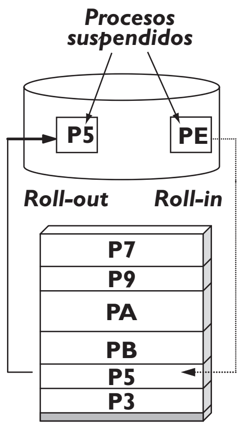


Figura 13.10. *Intercambio memoria principal disco. Se saca P5 para introducir PE.*

Para evitar esta limitación existe un procedimiento consistente en transvasar a disco (hacia afuera, *roll-out*) un proceso, *P5* en la Figura 13.10, de los que están residiendo en memoria, para dar cabida a otro, *PE*, que se transvasa de disco a memoria (hacia adentro, *roll-in*). El proceso *P5* será vuelto a cargar en memoria para cuando le llegue su turno. Este procedimiento, denominado **intercambio memoria principal/disco** (*swapping*) es gestionado por un módulo del sistema operativo, denominado **planificador a medio plazo** o **intercambiador**, y permite al computador realizar concurrentemente un número de procesos que no esté limitado por la capacidad de la memoria principal. La eficiencia o velocidad de esta técnica depende de la velocidad de transferencia memoria/disco y de la velocidad de funcionamiento de la unidad de disco, necesitándose disponer de DMA y/o procesadores de E/S.

EJEMPLO 13.11

Supóngase que un computador que ejecuta 10^6 inst./seg., de longitud de palabra de 32 bits, dispone de una unidad de disco con un tiempo medio de acceso de 14 ms, y tasa de datos de 4 MB/s. Hacer una estimación del tiempo que se tarda en transvasar de memoria a disco un proceso de 1 Mpalabras.

$$1Mp = 1Mp \times 4\frac{p}{B} = 4\ MB$$

$$t = 14 \cdot 10^{-3} + \frac{4\,MB}{4\,MB/s} = 1{,}014\ s$$

En este intervalo de tiempo el procesador puede ejecutar:

$$N = 1{,}014\,s \times 10^6\ \frac{inst}{s} = 1.014.000\ instrucciones$$

Obsérvese que este tiempo es comparable a diez quanta de multiprogramación (del orden de décimas de segundo); ahora bien mientras actúa el controlador DMA, el procesador podría seguir trabajando con uno de los procesos preparados en memoria, mientras se efectúan las transferencias entre memoria y disco (véase Sección 9.5.4.3 y 12.5). En consecuencia, el procesador no estaría detenido en una gran parte del tiempo indicado.

Con la técnica de intercambio el número máximo de procesos en ejecución no viene impuesto por la capacidad de memoria; ahora bien, si la relación:

$$N_{máx,\,procesos} = \frac{\text{capacidad de memoria}}{\text{número de procesos}}$$

es baja, el funcionamiento del sistema puede ser muy lento, ya que el intervalo de tiempo entre dos turnos sucesivos a un proceso dado puede ser relativamente grande. Este problema puede reducirse fácilmente con una ampliación de memoria principal adecuadamente dimensionada.

Los procesos transvasados a disco se dice que están en **estado intercambiado**.

13.5.5 Estados de un proceso

Un proceso, desde el punto de vista de su ejecución, puede estar en una de diversas situaciones o estados, cuyo número y denominación depende del sistema operativo. Un posible

modelo de siete estados se incluye en la Figura 13.11. La mayoría de éstos han sido analizados anteriormente, por lo que los resumiremos muy brevemente.

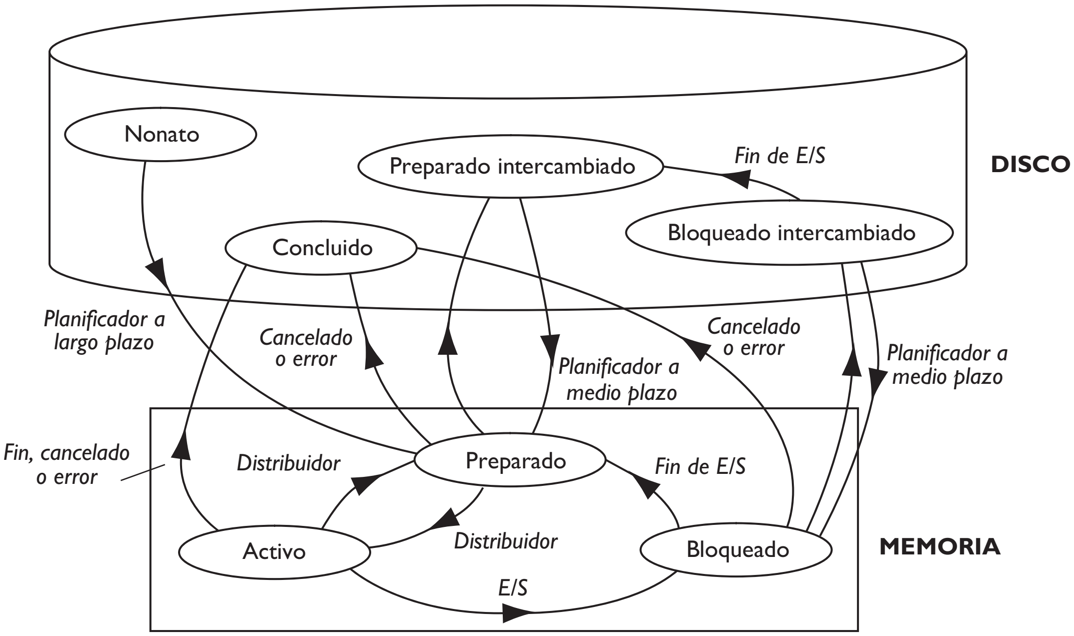


Figura 13.11. *Planificación con un modelo de siete estados.*

Un **proceso nonato** es un programa que no ha iniciado su ejecución; es decir, en memoria o disco pendiente de que se inicie su ejecución. Puede ser, por ejemplo, un programa retenido en una cola esperando a que el **planificador a largo plazo** (de trabajos) lo seleccione, o un programa en disco que puede ser seleccionado por medio de un icono para ejecutar.

Recordemos que un proceso está en **estado preparado** cuando se encuentra en memoria principal, sin operaciones de E/S pendientes, y apto para entrar (o continuar) en ejecución en el instante que el **planificador a corto plazo (distribuidor)** le asigne el procesador. El **estado activo** corresponde al proceso que en ese momento está siendo atendido por el procesador. Un proceso activo sale del estado de ejecución, en un sistema de tiempo compartido, cuando pasa a **estado de bloqueado** o el distribuidor lo interrumpe para atender a otro de mayor prioridad (o que le corresponde el turno).

Un proceso pasa del estado de bloqueado al estado preparado cuando acaba la operación de E/S pendiente. También en un sistema en que se gestiona la memoria con intercambio, y con más procesos en ejecución concurrente de los que admite la memoria principal, un proceso bloqueado puede ser transvasado a disco (**estado bloqueado intercambiado**). Si la carga de procesos es grande, incluso los procesos preparados pueden transvasarse a disco, pasando así al estado de **preparado intercambiado**. El **planificador a medio plazo** se encarga de seleccionar el momento en los que un proceso suspendido debe pasar de disco a memoria principal

Cuando desde un terminal interactivo se *abre línea* con el computador o se le da la vez a un programa de la cola, el proceso correspondiente entra en estado preparado, desplazándose

por los estados del diagrama de la Figura 13.11. Cuando finaliza la ejecución de un proceso o se detecta un error grave en él, el proceso pasa a **estado de concluido***,* liberando el sistema operativo la zona de memoria que ocupaba el proceso.

Los procesos más importantes, mientras progresa su ejecución, pueden ser obligados a residir en memoria principal. Éstos se dice que son **procesos residentes** o *no intercambiables*. Tal es el caso del núcleo y otros módulos importantes del sistema operativo. A veces parte del sistema operativo se encuentra grabado en la memoria ROM, y en los sistemas embebidos suele estarlo en su totalidad.

13.6 Gestión de la memoria

Un programa máquina es una secuencia de instrucciones en código máquina. Estas instrucciones, en el momento de ejecutarse "encajan" en palabras de memoria que pueden numerarse correlativamente de la *0* a la $n - 1$ (suponiendo que el programa ocupa *n* palabras de memoria). Estas direcciones se denominan **direcciones virtuales** o **lógicas**, *dv*.

El programa anterior, al ser cargado en memoria, ocupará *n* determinadas posiciones de la misma. Supongamos que las instrucciones del programa se almacenan consecutivamente; si se cargan a partir de la dirección *dB*, el programa quedará ubicado de la dirección *dB* a la $dB + (n - 1)$ (Figura 13.12). A *dB* se le suele denominar **dirección base** y a la direcciones *df* en que realmente se almacena las instrucción de la **dirección virtual** *dv*, se le denomina **dirección física**. Se verifica que:

$$df = dB + dv \qquad para\ todo \quad 0 \le dv \le n-1$$

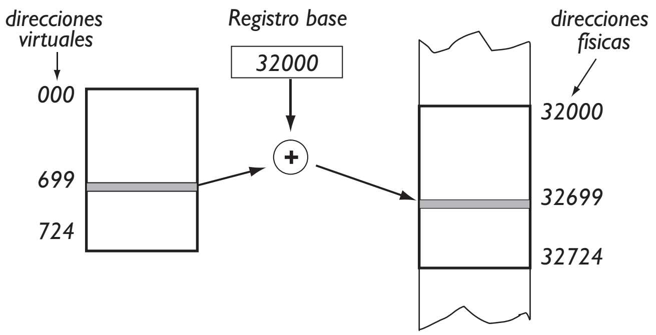


Figura 13.12. *Transformación de direcciones virtuales en direcciones físicas.*

A veces el programa se considera por el sistema operativo dividido en segmentos. Un **segmento** es un grupo lógico de información, tal como un programa **(segmento de código)**, una pila asociada al programa **(segmento de pila)**, o un conjunto de datos asociados al programa **(segmento de datos)**. En consecuencia, el tamaño de cada segmento de un programa no está establecido a priori y depende totalmente del programa de que se trate. Un programa ejecutable (listo para ser ejecutado por el procesador) es una colección de segmentos.

En los sistemas operativos de monoprogramación, la memoria principal se puede organizar de diversas maneras, según indica Figura 13.13. El sistema operativo puede ocupar las primeras posiciones de memoria o las últimas (que pueden ser de tipo ROM), o incluso parte

del sistema operativo puede estar en la zona RAM de direcciones bajas, y otra parte de él (los gestores de periféricos) y el cargador inicial en ROM, en las direcciones altas. Esta última situación se corresponde con la de los PC compatibles, en los que la ROM se denomina **Sistema Básico de Entrada/Salida** o **BIOS** (*Basic Input Output System*). En cualquier caso, cuando el usuario da una orden, si el proceso que la implementa no está en memoria, el sistema operativo se encarga de cargarlo en memoria desde disco, y el distribuidor da paso a su ejecución.

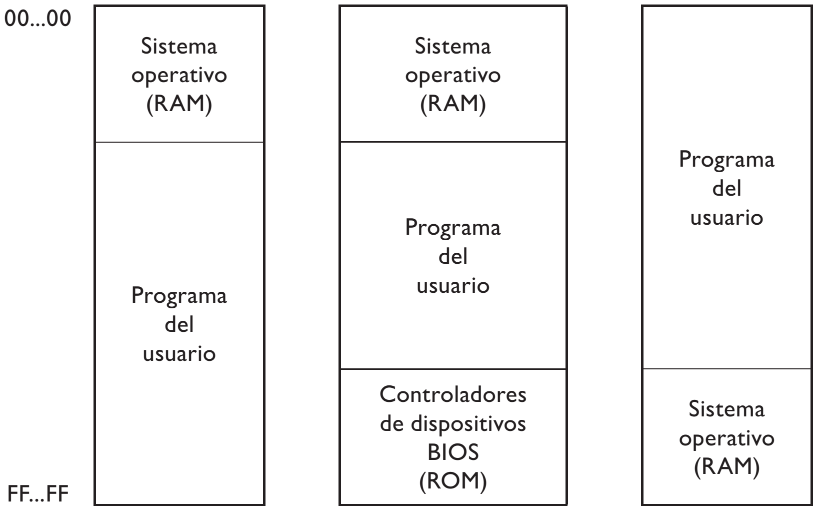


Figura 13.13. *Distintas alternativas de ubicación del sistema operativo en la memoria principal.*

En un sistema de multiprogramación, el sistema operativo, de acuerdo con los espacios libres de la memoria, asigna una dirección base a cada programa, y lo carga en la forma que se indicó con anterioridad (Figura 13.13). Las transformaciones entre direcciones virtuales y físicas antiguamente se realizaba por software, pero en la actualidad se suele realizar con ayuda de un registro base (donde se carga previamente la dirección base) que se encuentra o en el propio procesador o en unos circuitos específicos que forman parte de una **Unidad de Gestión de Memoria** (**MMU**, *Memory Management Unit*), que se integra en el mismo chip del microprocesador (Figura 13.14).

La asignación de memoria para distintos procesos ejecutándose concurrentemente suele hacerse, dependiendo del sistema operativo, de alguna de las siguientes formas:

- Particiones estáticas
- Particiones dinámicas
- Segmentación
- Paginación
- Memoria virtual

que se analizan en las siguientes secciones.

La memoria virtual es un procedimiento que asigna memoria y además posibilita que puedan ejecutarse programas con una capacidad mayor que la capacidad real de la memoria física disponible.

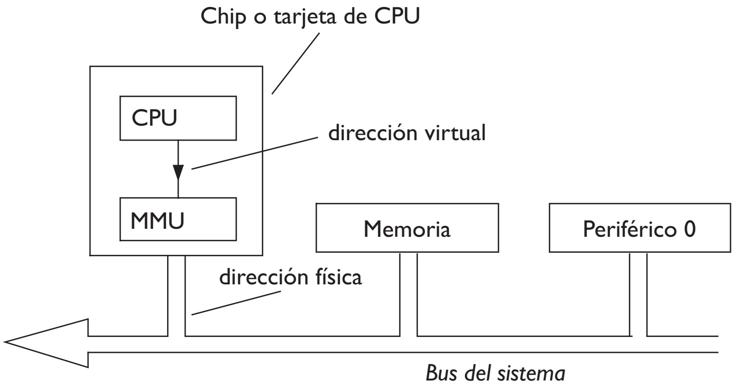


Figura 13.14. *La MMU transforma direcciones virtuales en direcciones físicas.*

13.6.1 Particiones estáticas

La memoria se divide en cierto número de **particiones** o zonas, cada una de las cuales contendrá un proceso En la Figura 13.15 se incluye un ejemplo de un sistema con una memoria principal de 2 MB y 8 particiones constantes de 256 KB. Las direcciones de base son las direcciones de comienzo de cada partición. El tamaño de la partición es determinado por el operador o por el sistema operativo. Tamaños usuales son: 8 K, 16 K, 32 K, ó 64 K. El sistema operativo mantiene una tabla (Tabla 13.3) en la que cada fila corresponde a una partición, conteniendo la posición base de la partición, su tamaño (no todas las particiones tienen por qué ser iguales) y el estado de la partición (ocupada o no ocupada). El planificador de trabajos, una vez que una partición está libre, hace que se introduzca el programa de máxima prioridad que haya en la cola de espera y que quepa en dicha partición.

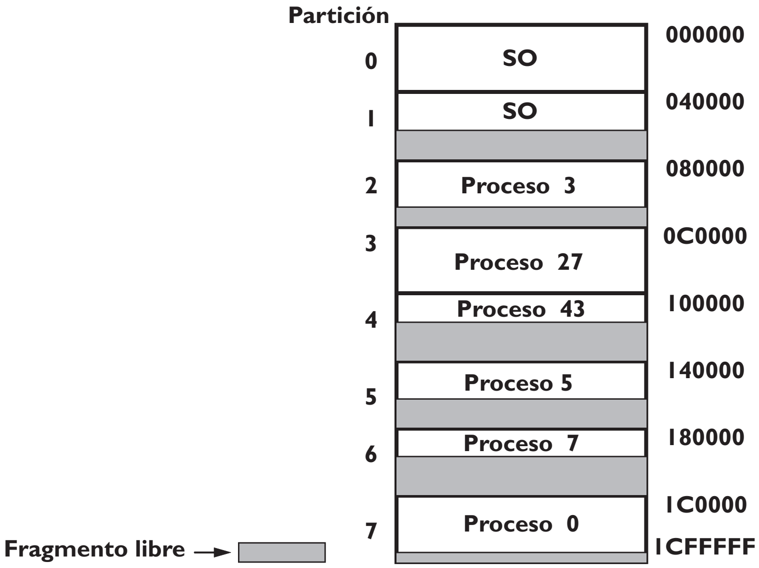


Figura 13.15. *Asignación de memoria con particiones estáticas.*

Si el espacio de una partición es m palabras y el programa ocupa n posiciones, se verificará siempre que:

$$n \leq m$$

Cada partición tendrá unas posiciones no utilizadas si $n < m$. Se denomina *fragmentación interna* o **fragmentación de una partición** a:

$$g = m - n$$

Tabla 13.3. *Tabla de particiones correspondiente al ejemplo de la Figura 13.15.*

Partición	Dirección base	Tamaño partición	Proceso
0	000000	256 k	SOa
1	040000	256 k	SOb
2	080000	256 k	P3
3	0C0000	256 k	P27
4	100000	256 k	P43
5	140000	256 k	P5
6	180000	256 k	P7
7	0C0000	256 k	P0

13.6.2 Particiones dinámicas

Los programas, Figura 13.16a, son introducidos por el sistema operativo inicialmente en posiciones consecutivas de memoria, no existiendo por tanto particiones predefinidas.

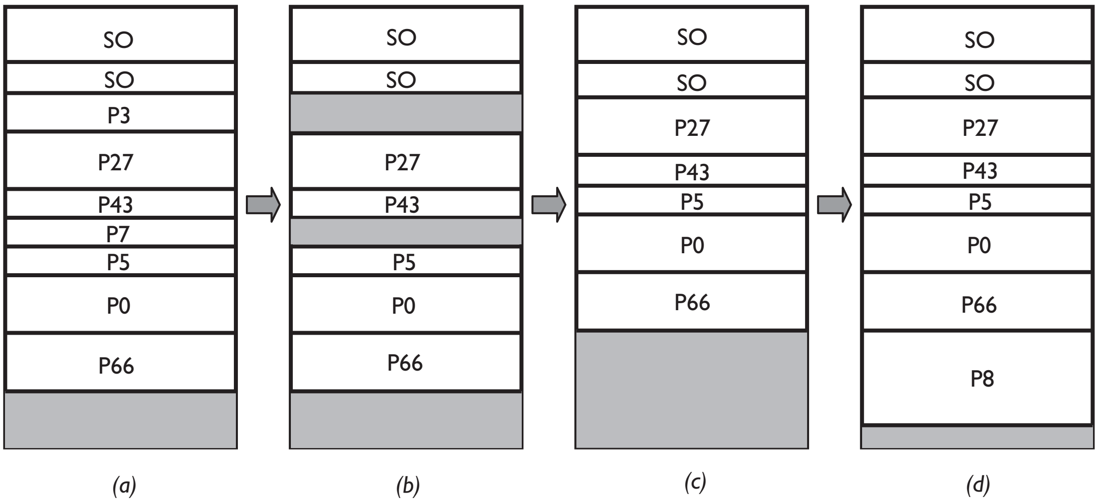


Figura 13.16. *Asignación dinámica: (a) los procesos se ubican correlativamente; (b) P3 y P7 acaban; (c) compactación; (d) entra P8.*

El sistema operativo puede gestionar el espacio de memoria usando una **tabla de procesos** (Tabla 13.4a), en la que cada línea contiene el número de proceso o identificativo del mismo,

el espacio que ocupa, y la dirección base. Existe una tabla complementaria a la anterior con los fragmentos o huecos libres (Tabla 13.4b). El planificador de trabajos, periódicamente consulta la tabla, introduciendo en memoria los programas que quepan en los fragmentos. Obviamente, al iniciarse una sesión de trabajo se carga el primer programa, dejando un fragmento libre donde se pueden incluir otros programas. Al ir acabando de ejecutarse los procesos (Figura 13.16b), el número de fragmentos crecerá (podrá llegar a haber tantos como procesos hayan pasado por memoria) y el espacio de cada uno de ellos disminuirá, llegando un momento en que el porcentaje de memoria aprovechado es muy reducido. El problema puede resolverse haciendo una **compactación** (Figura 13.16c), como, por ejemplo, realizaban los computadores DG Eclipse, CDC 6600 o la PDP 11. Ésta consiste en agrupar todos los fragmentos cuando acaba la ejecución de un proceso, quedando así sólo uno grande. La compactación se efectúa cambiando de sitio o **reubicando** los procesos en ejecución.

EJEMPLO 13.12

En la Figura 13.16b se muestra la memoria cuando acaban los procesos P3 y P7; pueden ahora introducirse tres programas: uno de 180 KB o menor, otro de 128 KB o menor, y otro en el fragmento inicial libre de 588 KB. Si en la cola de espera sólo hay programas mayores de 588 KB, no entrará ninguno. En la Figura 13.16c se muestra cómo queda la memoria después de una compactación; ahora caben uno o varios procesos, con una capacidad total no mayor de 896 KB. En la Figura 13.16d entra el programa P8, que ocupa 650 KB.

Tabla 13.4. *Tabla de procesos y fragmentos libres para el ejemplo de la Figura 13.16a.*

Tabla de procesos		
Proceso	**Dirección base**	**Tamaño**
SOa	000000	256 kp
SOb	040000	128 kp
P3	060000	180 kp
P27	08D000	256 kp
P43	0CD000	128 kp
P7	0ED000	128 kp
P5	10D000	128 kp
P66	12D000	256 kp

(a)

Tabla de fragmentos libres		
Fragmento	**Dirección base**	**Tamaño**
0	16D000	588 K

(b)

13.6.3 Segmentación

El programa se considera dividido en sus segmentos (recordar que los segmentos los define el programador, según se vio al comienzo de la Sección 13.6). La gestión la realiza el sistema operativo, como con las particiones dinámicas (Figura 13.16 y Tabla 13.4), sólo que cada partición no corresponde a un proceso sino a un segmento. El sistema operativo mantiene una tabla-mapa de segmentos, indicando la ubicación en memoria de cada uno de ellos y su tamaño.

Como el tamaño de los segmentos es variable, antes de realizar un acceso a memoria se comprueba que el desplazamiento no supere al tamaño del segmento, para proteger a las zonas de memoria ocupadas por otro segmento (Figura 13.17), como se explica en el siguiente ejemplo.

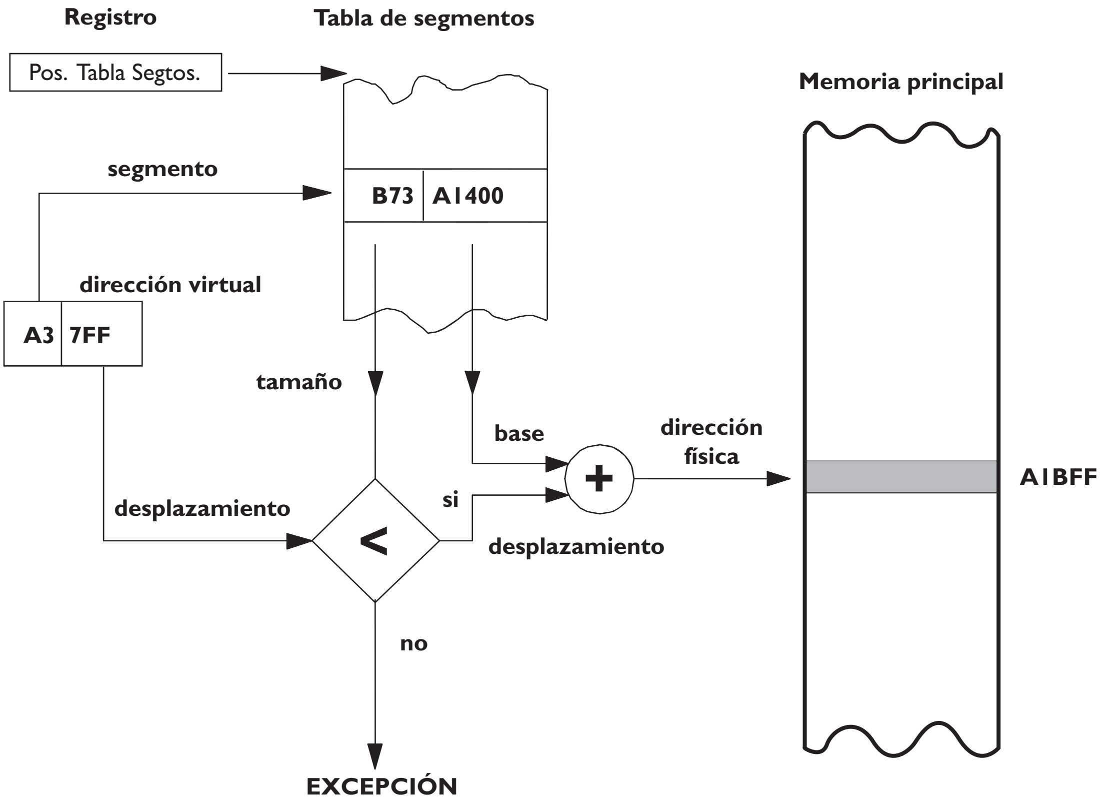


Figura 13.17. *Esquema de transformación de una dirección virtual en una dirección física en un sistema con segmentación.*

EJEMPLO 13.13

Supóngase un computador que admite hasta 256 segmentos de 4 KBytes cada uno como máximo. Las direcciones lógicas estarán compuestas de 20 bits (5 cifras hexadecimales), los 8 primeros bits (2 primeras cifras hexadecimales) identificarán el segmento, y los 12 bits restantes (3 cifras hexadecimales) identificarán el desplazamiento dentro del segmento. La dirección lógica A37FF se transforma en una dirección física según el esquema que muestra la Figura 13.17. Obsérvese que el tamaño del segmento es B73. Si por error se hubiese generado en este segmento (A3) una dirección lógica mayor o igual a A3B73, el sistema operativo generaría una excepción (interrupción interna) de "error de direccionamiento". Con la Tabla de segmentos se obtiene la dirección base de memoria donde se almacena el segmento A3, que resulta ser A1400, con lo que la dirección física correspondiente será:

$$A1400 + 7FF = A1BFF$$

La segmentación permite que ciertos procesos puedan compartir código (rutinas, etc.) o datos comunes sin necesidad de estar duplicados en memoria principal. Así, si seis usuarios están utilizando interactivamente un procesador de textos determinado, no sería necesario que estuviesen cargadas en memoria seis copias idénticas del código del programa procesador de textos. El código del programa estaría una sola vez, y el sistema operativo se limitaría a anotar en las tablas-mapas de segmentos de cada uno de los procesos la dirección donde se encuentra el código. Por tanto, en un momento dado, en memoria existirían: segmentos asignados a trabajos concretos, segmentos compartidos, y bloques libres.

EJEMPLO 13.14

Los compatibles PC, desde su inicio (microprocesador 8088), utilizan un direccionamiento con segmentación. También es corriente combinar la segmentación con la paginación, tal es el caso de los computadores IBM-370, GE-645, y del microprocesador 80386 en adelante.

13.6.4 Paginación

Con este procedimiento la memoria principal se estructura en **marcos de página** (también denominados **bloques de memoria**) de longitud fija: suelen ser de 512 bytes (DEC VAX, IBM AS/400), 4 KBytes (Pentium, PowerPC), 8 KBytes (DEC Alpha) o 4 MBytes. De esta forma, por ejemplo, la memoria de un computador de 512 MBytes podría quedar dividida en 8.192 marcos de página de 64 KBytes cada uno, o en 131.072 de 4 KBytes. Cada marco de página se identifica con un número correlativo (de 0 a FFFF, ó 0 a F, en los casos anteriores). Asimismo los procesos de los usuarios se dividen en zonas consecutivas, denominadas **páginas lógicas** o **virtuales** (o, simplemente, **páginas)**. Para un sistema dado, la capacidad de página y marco de página son coincidentes, de forma que cada página se almacena en un marco.

El fundamento de la paginación reside en que no es necesario que un proceso se almacene en posiciones consecutivas de memoria. Las páginas se almacenan en marcos de página libres independientemente de que estén o no contiguos.

EJEMPLO 13.15

En la Figura 13.18 se muestra como podría almacenarse un proceso de 18 KBytes en un sistema con páginas de 4 KBytes y memoria principal de 16 MBytes. El proceso tendrá 5 páginas (la última de ellas ocupada sólo con 2KBytes) y la memoria principal 4096 (en la figura sólo se representan las 13 primeras). Cada marco de página contiene instrucciones consecutivas.

Una dirección virtual (esto es, una dirección del programa) puede considerarse formada por el **número de página** (bits más significativos) y un **desplazamiento** (o posición relativa) dentro de la página (bits menos significativos). Para mejor comprender lo anterior, supongamos que estamos en base 10, y las páginas son de 100 posiciones. Entonces la dirección 328, corresponderá a la página 3, y, dentro de ella tendrá una posición relativa (desplazamiento) de 28 posiciones (Figura 13.19).

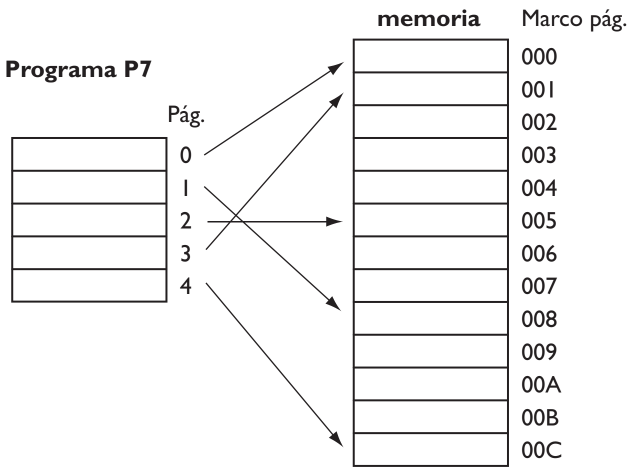


Figura 13.18. *Las páginas de un proceso se almacenan en marcos de página no necesariamente consecutivos.*

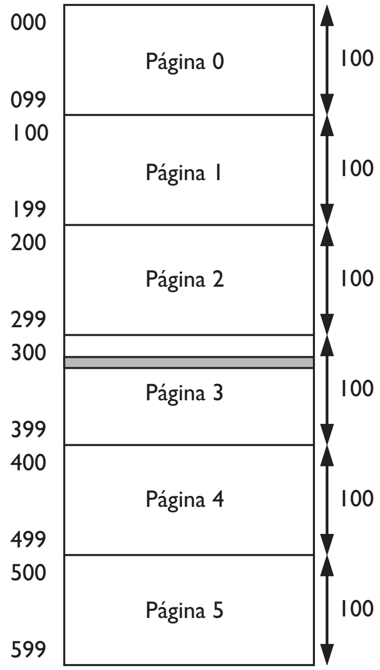


Figura 13.19. *Ejemplo de asignación de direcciones a los marcos de página, suponiendo una notación decimal.*

Cada proceso tiene asociado una Tabla de Páginas que, sencillamente, indica el marco de página donde se encuentra almacenada la página correspondiente. La Tabla de Páginas del Proceso *P7* de la Figura 13.18 se muestra en la Tabla 13.5. La MMU hace la correspondencia entre dirección virtual y física, y puede contener la tabla de páginas o un registro indicando la posición de memoria donde se encuentra (Figura 13.20).

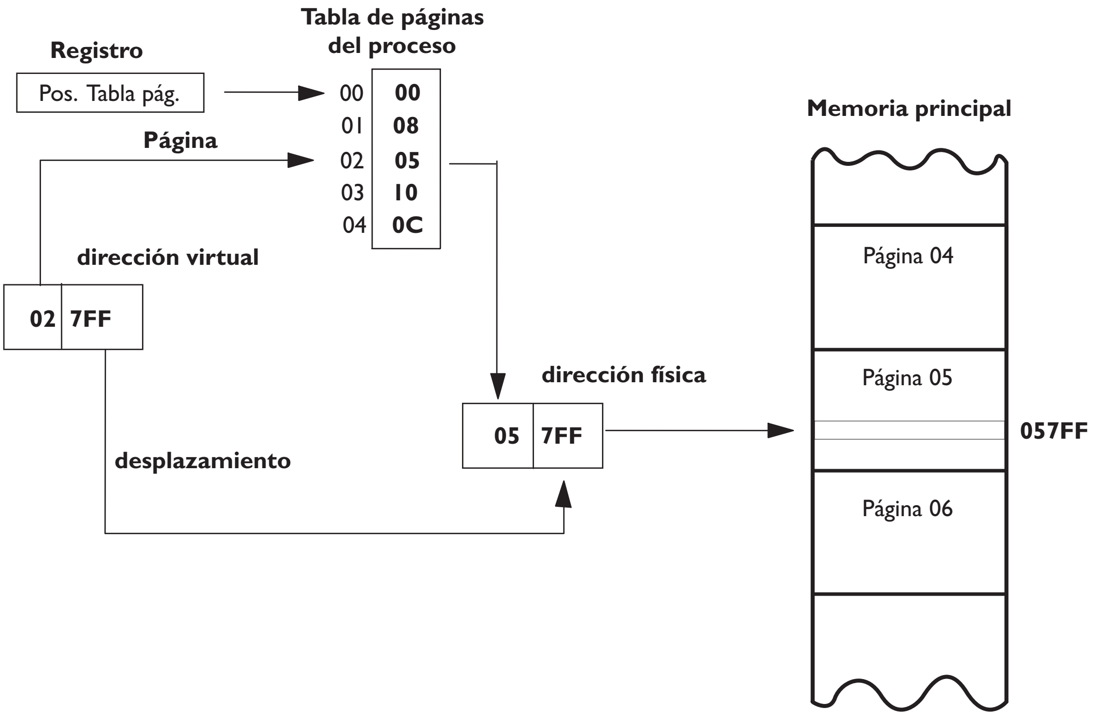


Figura 13.20. *Esquema de transformación de direcciones en un sistema con paginación.*

Tabla 13.5. *Ejemplo de una tabla de marcos de página (obsérvese la ubicación de las páginas de P7, en concordancia con la Figura 13.18).*

(marco de página)	**Contenido (Proceso, página)**	**Estado (0: ocupado, 1: libre)**
000	P7,0	0
001	P7,3	0
002	SO,3	0
003	SO,4	0
004	*(libre)*	1
005	P7,2	0
006	P5,0	0
007	P5,1	0
008	P7,1	0
009	P3,0	0
00A	*(libre)*	1
00B	*(libre)*	1
00C	P7,4	0
00D	P3,1	0
......	...	...
......	...	...
......	...	...
FFF	SO,2	0

En el bloque de control de cada proceso (PCB) se guarda la posición donde se encuentra almacenada su correspondiente tabla de páginas para tenerlo en cuenta en los cambios de contexto. El sistema operativo además mantiene una **tabla de marcos de página,** donde se especifica el proceso y página contenido en cada uno de los marcos y su estado (libre o ocupado). En la Tabla 13.5 se muestra un ejemplo.

13.6.5 Memoria virtual

La memoria virtual permite a los usuarios hacer programas de una capacidad muy superior a la que físicamente tiene el computador. Por ejemplo, el microprocesador Pentium admite hacer programas de hasta 64 Terabytes (2^{46}), utilizando una memoria principal mucho menor (por ejemplo de 512 MBytes). En realidad la memoria virtual hace posible que la capacidad máxima de los programas venga limitada por el espacio que se reserve en disco para ella y no por la memoria principal. En definitiva, los sistemas con memoria virtual presentan al usuario una memoria principal *aparente* mayor que la memoria física real (véase Figura 13.21).

Por otra parte, la memoria virtual permite que aumente el número de procesos en la memoria principal en ejecución concurrente, ya que con aquélla sólo es necesario que esté en memoria principal un trozo mínimo (una página) de cada proceso, y no el proceso completo.

Para implantar la memoria virtual suele utilizarse una de las siguientes técnicas de gestión:

- Gestión de memoria por páginas.
- Gestión de memoria segmentada.
- Gestión de memoria segmentada-paginada.

La memoria virtual se basa en que, según ya se comentó en la Sección 10.1, las instrucciones de un programa que se ejecutan sucesivamente (en un corto intervalo de tiempo) están en direcciones muy próximas (**principio de localidad temporal**), y en que los programas suelen estar redactados con gran linealidad; es decir, no suelen abundar los saltos entre posiciones de memoria distantes (**principio de localidad espacial**).

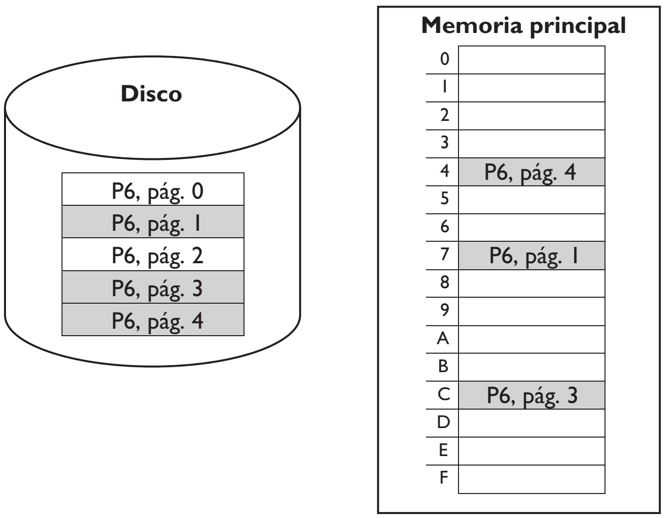


Figura 13.21. *Con la memoria virtual se mantiene siempre una copia completa del proceso en disco, y sólo algunas de sus páginas (las que aparecen en la figura sombreadas) en la memoria principal.*

En un sistema de memoria virtual (Figura 13.21) se mantiene en disco un archivo con la **imagen del proceso** completo, que está troceado en páginas o segmentos, dependiendo de la técnica de gestión que se use. En cambio en la memoria principal únicamente se carga la página (o segmento, en su caso) que en ese momento deba estar en ejecución, intercambiándose páginas entre disco y memoria principal cuando sea necesario.

La gestión de memoria virtual segmentada es más compleja que la de paginación, ya que los segmentos son de tamaño variable y son más difíciles de gestionar; las páginas, por el contrario, son de capacidad constante y preestablecida. Por ello lo habitual es utilizar gestión de memoria por paginación, o por segmentos paginados.

La memoria virtual con paginación combina las técnicas de paginación (Sección 13.6.3) e intercambio (Sección 13.5.3). Por lo general se utiliza un método de **intercambio perezoso** (*lazzy swapper*): únicamente se lleva a memoria una página cuando es necesaria para algún proceso (**demanda de página**); de esta forma, al utilizar en un momento dado cada proceso menos memoria, el número de procesos en ejecución concurrente puede aumentar.

La gestión de la memoria virtual con demanda de páginas la puede hacer el sistema operativo con ayuda de dos tipos de tablas:

- **Tabla de páginas del proceso**. El sistema operativo crea y mantiene una de estas tablas por cada proceso. En la Tabla 13.6 se incluye un ejemplo de tabla de páginas de un determinado proceso (Proceso *P6*). Cada fila corresponde a una de las páginas del proceso, y contiene los siguientes campos:
  - *Dirección de disco donde comienza la página,* que puede ser el número de la unidad de ubicación (véase Sección 13.8.1) del disco donde se encuentra esa página en el proceso imagen.
  - *Marco de memoria principal* en el que está (en su caso) la página, y un bit que indica la *situación de la página* (por ejemplo, 0 si está dentro de la memoria principal, ó 1 si está sólo en memoria secundaria, fuera de la memoria principal).

Tabla 13.6. *Ejemplo de tabla de páginas de un proceso (P6) en un sistema de MV.*

(página)	**Posición en disco (unidad de ubicación)**	**Posición en memoria (marco de página)**	**Ubicación (0: memoria, 1: sólo disco)**
0	7ABC	-	1
1	CA73	7	0
2	4BC9	-	1
3	573C	C	0
4	A340	4	1

- **Tabla de marcos de páginas** (Tabla 13.7). Cada fila de esta tabla corresponde a un marco de página de memoria principal, y contiene lo siguiente:
  - *(proceso, página)* que está en el marco de página.
  - *Estado del marco de página* (0: ocupado; 1:libre, por ejemplo).
  - *Modificación*; es un bit, M, para indicar si, desde el instante que se ha cargado la página desde el disco por última vez se ha modificado (1) o no (0) dicha página. Obsérvese que si M = 0 quiere decir que la página almacenada en la memoria principal coincide exactamente con la página almacenada en disco.
  - *Información adicional:* depende del algoritmo de reemplazo que se utilice, como más adelante se verá.

Tabla 13.7. *Ejemplo de un trozo de tabla de marcos de página en un sistema de MV.*

(marco)	Contenido (Proceso, página)	Estado (0: ocupado, 1: libre)	Modificación (0: no modificado, 1: modificado)
000	SO,5	0	1
001	SO,6	0	1
002	SO,7	0	1
003	P2,5	0	0
004	P6,04	0	0
005	P2,4	0	0
006	*(libre)*	1	0
007	P6,1	0	0
008	P8,0	0	0
009	P2,5	0	1
00A	P4,5	0	0
00B	P4,6	0	0
00C	P6,03	0	1
00D	P4,3	0	0
......	...	...	...
......	...	...	...
......	...	...	...
FFF	SO,4	0	0

Supongamos que el proceso *P6* hace referencia a una instrucción o dato de la página 0. El sistema operativo consulta la Tabla de Páginas P6 y detecta que no está en memoria principal (bit ubicación = 1). En esta situación se dice que se ha producido una excepción de **fallo de página**. El sistema operativo ahora busca en la tabla de marcos de página en el campo *estado* si hay un marco libre, y en nuestro caso encuentra que el 6 lo está (*estado* = *1*). Entonces con ayuda del DMA se carga rápidamente la Página 0 solicitada, que según la tabla de páginas se encuentra en disco a partir de la posición *7ABC*. El sistema operativo actualiza las tablas de acuerdo con la nueva situación: en la tabla de páginas de *P6* fila 0 hace *ubicación* igual a 0 y *posición* igual a *6*; y en la tabla de marcos de página, en la fila *6* primer campo coloca *6,0*; y *estado* y *modificación* se hacen igual a cero.

Imaginemos que posteriormente se hace una referencia a la Página *2* que tampoco se encuentra en memoria principal. Ahora no hay ningún marco libre, por lo que el sistema operativo (con un **algoritmo de reemplazo de página**) debe eliminar de la memoria principal una de las páginas para introducir la solicitada. Supongamos que el algoritmo indica que debe sustituirse el contenido del marco de página 004 constituido por *P6,04*. Como su bit *modificación* es 0, quiere decir que esta página no se ha modificado desde que se introdujo en la memoria principal, por lo que está exactamente igual en disco. El sistema operativo, sin más, carga en ese marco la página solicitada (02), destruyendo la página previa (esto no causa problema por estar en disco). Si por el contrario hubiese que reemplazar la página 6,03 (en el marco de página 00C), como su bit *modificación* es 1, antes de cargar la nueva página, debe actualizarse la copia en disco de dicha página (que está a partir de la posición *573C*).

En la Figura 13.22 se incluye un esquema que indica la forma en que una dirección virtual, *27AB* de un proceso *P3,* se transforma en su dirección física, *FE7AB,* en un sistema operativo de memoria virtual.

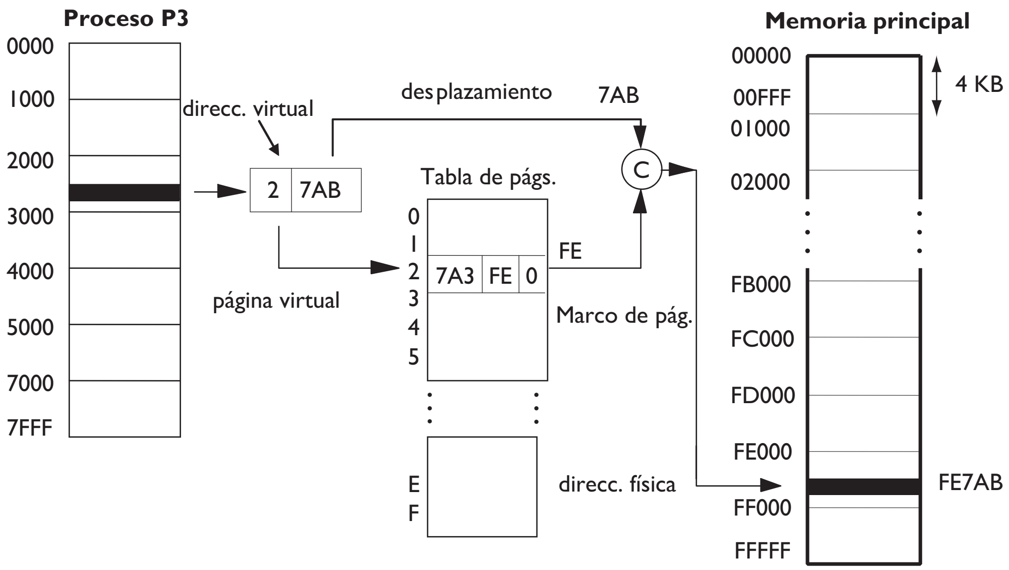


Figura 13.22. *Esquema de transformación de una dirección virtual en una dirección física en un sistema de memoria virtual.*

En relación con las páginas que deben estar en memoria hay que decidir dos cuestiones:

1. *Qué páginas deben cargarse en memoria.* Suele utilizarse una de las dos siguientes alternativas: **Paginación por demanda**: una página sólo se lleva a memoria cuando se hace referencia (*se accede*) a una dirección de ella, o **asignación previa**, se cargan en memoria más páginas de las referenciadas, usualmente consecutivas (para así aprovechar la localidad espacial, y para ganar en velocidad: se transmite más rápidamente un bloque de 8 KB que dos de 4 KB).
2. *Qué página debe sustituirse en caso de producirse un fallo de página.* La búsqueda de algoritmos eficientes para decidir esta cuestión es uno de los temas de mayor interés en el diseño de sistemas de memoria virtual ya que cada fallo de página supone una pérdida de tiempo del procesador que puede hacer disminuir considerablemente el rendimiento del computador incluso aunque toda la gestión se realice con hardware específico. Usualmente cuando se produce un fallo de página en un proceso, se sustituye una página del propio proceso, **sustitución con alcance local**, aunque a veces se realiza una **sustitución con alcance global**; es decir, se analizan todas las páginas de memoria para realizar la sustitución.

A continuación se describen algunos de los algoritmos de sustitución más difundidos.

- **LRU** (*Least Recently Used*): se sustituye la página que lleve mas tiempo sin usar. Se basa en el principio de localidad temporal ya que supone que la página utilizada hace mayor tiempo, es menos probable que se use próximamente. Una forma de implementar este algoritmo consiste en disponer de un contador (de 64 bits, por ejemplo) que se incrementa automáticamente al ejecutar cualquier instrucción, y añadir a la tabla de marcos de página una nueva columna, C (también de 64 bits). Cuando se hace una referencia a un marco de página, se actualiza su valor de C. Al producirse un fallo de página el sistema operativo elimina la página que esté en el marco con menor valor

de C. Esta técnica resulta costosa de implementar: en cada referencia a memoria hay que actualizar el valor de C correspondiente.

- **FIFO** (*First In First Out*): se sustituye la que lleve más tiempo en memoria. Se puede implementar, o bien con una estructura de tipo cola donde se vayan incluyendo sucesivamente en una cola los códigos de los marcos que se van cargando; se sustituye la página del marco primero de la cola y su referencia se elimina de la cabecera de la cola. También puede hacerse como LRU, pero actualizando el campo C en el momento de cargar la página en el marco correspondiente (en lugar de cuando se referencia).
- **NRU** (*Not Recently Used):* se sustituye una página no utilizada recientemente. Puede implementarse añadiendo en la tabla de marcos de página, en lugar del campo C, otro campo, *referido* (R), pero de un solo bit. En cada interrupción de reloj (usualmente cada 20 milisegundos) se hacen cero los bits R de todos los marcos, y si se hace referencia a un marco se pone su bit R a uno. De esta forma los marcos de página que en un momento dado tengan 0 no han sido referenciados desde la última interrupción de reloj, y las páginas almacenadas en ellos son candidatas a ser sustituidas caso de producirse un fallo de página. El sistema operativo para hacer la sustitución analiza los bits R (referido) y M (modificado), de forma que la prioridad de sustitución, de mayor a menor, es la siguiente:

$$R = 0, M = 0$$
$$R = 0, M = 1$$
$$R = 1, M = 0$$
$$R = 1, M = 1$$

Si hay varios marcos de página en iguales condiciones para ser sustituidos, se elige aleatoriamente uno de ellos.

- **RELOJ**: se sustituye la página más antigua no utilizada recientemente. Para implementar esta técnica se utilizan los bits R (referido), y un puntero que recorre circularmente los marcos de página (del proceso, si el alcance de sustitución es local) y que se coloca inicialmente en la página que primero se cargó en memoria. En resumen el comportamiento del algoritmo es como sigue:
  - Cuando se introduce una página, el bit de referencia del marco se pone a cero ($R = 0$).
  - Si se accede a un marco, se hace $R = 1$, y no se avanza el puntero.
  - Si se produce un fallo de página:
    - Se analiza el valor de R del marco al que apunta el puntero:
      - Si $R = 0$, se reemplaza la página y se avanza una posición el puntero.
      - Si $R = 1$, se cambia $R = 0$, y así sucesivamente se avanza el puntero hasta encontrar un marco con $R = 0$, en donde se reemplazó la página.

Puede observarse que este algoritmo es muy similar al FIFO, pero si una página antigua ha sido referenciada ($R = 1$) no se sustituye.

13.6.6 Gestión de memoria en el PENTIUM

Consideramos como ejemplo de sistema de gestión de memoria el Pentium, debido a que con él se tiene un ejemplo muy completo por admitir segmentación pura (compatible con el 80286), paginación y segmentación paginada. En la Figura 13.23 se muestra un esquema simplificado de la misma.

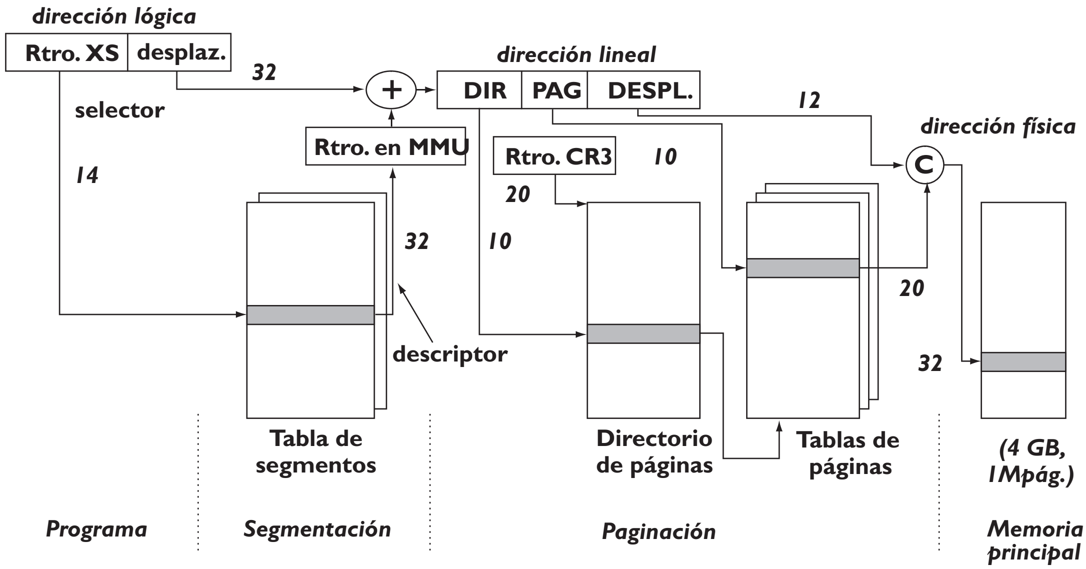


Figura 13.23. *Esquema simplificado de la gestión de memoria en el Pentium II.*

La segmentación se fundamenta en dos tablas: la de **descriptores globales** (GDT, *Global Descriptor Table*) y la de **descriptores locales** (LDT, *Local Descriptor Table*). La primera es común a todos los programas y contiene una descripción de los segmentos asociados con el sistema operativo. Por el contrario cada programa tiene su propia LDT, con descripción de sus segmentos particulares: de código, de pila, de datos, etc. Cada elemento de las tablas de descriptores es un **descriptor de segmento** y contiene su dirección base (32 bits) y el límite del segmento (que puede llegar a tener una capacidad de 2^{32} bytes).

Cuando se ejecuta un programa en el Pentium se cargan en sus registros de segmento *CS* (código), *DS* (datos), *SS* (pila), *ES* (extra), etc. (*XS* en la figura) un valor de 16 bits denominado **selector** que indica (1 bit) si se hace referencia a la GDT o LDT del programa, y un segmento o posición (con 13 bits) de la tabla correspondiente.

Los descriptores seleccionados se cargan en registros de la MMU. Allí se suma la dirección base del segmento con el desplazamiento (dirección relativa) a la que se trata de acceder, obteniéndose una **dirección lineal** de 32 bits.

Si el Pentium está funcionando en modo de segmentación pura, la dirección lineal es la dirección física; si no es así, la dirección lineal se considera formada por tres campos:

1) **DIR** (10 bits), especifica un elemento del directorio (índice) de páginas (1.024 elementos de 32 bits), que es una tabla cuya posición inicial en memoria se encuentra en el registro CR3 interior al Pentium. Cada elemento del directorio hace referencia a la dirección donde se encuentra una tabla de páginas.
2) **PAG** (10 bits), especifica un elemento de la tabla de páginas (1.024 elementos de 32 bits) especificada por DIR y que contiene (20 bits) el código marco de página al que se accede.
3) **Desplazamiento** (12 bits), posición relativa dentro del marco de página referenciado.

De esta forma, concatenando los 20 bits del marco de página y los 12 bits del desplazamiento se obtiene la dirección física de 32 bits.

Para mejorar los tiempos de acceso y aprovechando el principio de localidad temporal en la MMU se almacenan los valores DIR-PAG más recientemente usados.

13.7 Gestión de entradas/salidas

El sistema operativo contiene los módulos software que comunican los programas con los controladores físicos que generan las señales de control para los periféricos. En definitiva el sistema de gestión de entradas/salidas traduce peticiones generales (abstractas) en operaciones concretas.

Los objetivos fundamentales del software de E/S del sistema operativo son:

1) Lograr que los periféricos se utilicen con *eficiencia*. Para ello, el sistema operativo puede gestionar los dispositivos de E/S considerándolos de una de las tres formas siguientes:
   a) **Dispositivos de uso exclusivo**. Consiste en asignar ciertos dispositivos (un teclado, un ratón, una pantalla, o una impresora; por ejemplo) a un proceso durante la duración del trabajo.
   b) **Dispositivos compartidos**. Son los dispositivos que pueden compartirse concurrentemente por varios procesos (discos o cintas magnéticas), en un servidor de archivos de una red LAN, por ejemplo. Si dos o más procesos requieren a la vez la misma unidad de disco, por ejemplo, el sistema operativo debe evitar los conflictos que pudiesen plantearse.
   c) **Dispositivos virtuales**. Se consigue que dispositivos en principio asignables (impresora, por ejemplo) puedan compartirse, optimizándose el rendimiento del sistema.
2) Que desde el punto de vista del nivel de máquina simbólica (usuarios y aplicaciones) las operaciones de E/S sean lo más generales posibles; es decir se puedan programar con *independencia del dispositivo*, transparentemente a las características particulares del hardware que se utiliza. Esto permite hacer programas que utilicen dispositivos y archivos de forma abstracta, sin necesidad de particularizarlos para los dispositivos donde estén almacenados dichos archivos (disquetes, discos duros, cintas DAT, etc.).

La independencia del dispositivo implica que se pueda asociar a cada uno de ellos un nombre simbólico, siendo las reglas de denominación uniformes para todos. En UNIX y en Windows NT, por ejemplo, los dispositivos se referencian como si fuesen archivos (especiales). Así en UNIX para leer de un periférico existe una llamada al sistema que expresada en lenguaje C tiene la siguiente forma:

numero = read (descriptorarchivo , zona , numB);

con la que se solicita al sistema operativo leer un número determinado de bytes (*numB*) (un ***registro lógico***) de un periférico (cuyo nombre simbólico se da por medio de *descriptorarchivo*), y se memorizan en una zona específica de memoria (*zona*) en donde el programa accederá a los datos solicitados. En la variable *numero* se almacena el número de bytes leídos, de forma que si no se pueden leer los *numB* bytes, *numero* contendrá el número real de bytes leídos (si se produce algún error, el sistema operativo hace *numero* = –1). Esta sencilla instrucción permite al usuario hacer abstracción del periférico donde se encuentra el archivo y, dentro de él, el registro. Además, en el nivel de máquina simbólica no tienen que programarse los complejos detalles requeridos por el hardware para realizar la lectura: valores iniciales del

DMA, asignación de zona intermedia de la memoria principal, leer estado del DMA al final de la operación, etc. (Sección 7.4). En lugar de ello, la instrucción citada realiza una llamada al sistema operativo, de forma transparente al usuario.

Se comprende mejor cómo actúa el software de E/S utilizando un modelo conceptual por capas (Figura 13.24). Las tres capas intermedias corresponden al sistema operativo. El nivel inferior del sistema operativo (controlador software) suele consistir en una biblioteca de rutinas de bajo nivel que trata directamente con el periférico o, más concretamente, con el **controlador *(hardware)* del dispositivo** que es quien realmente ejecuta la operación de E/S (en los puertos de entrada/salida, controladores DMA, canales, procesadores de E/S), y el nivel superior (software de E/S independiente del dispositivo), en contacto con los procesos de los usuarios, trata con las funciones de E/S de una forma general y desde un punto de vista lógico. De esta forma se logra que los parámetros del hardware afecten lo menos posible al software de E/S. Resumidamente las funciones de cada nivel son las que se indican a continuación.

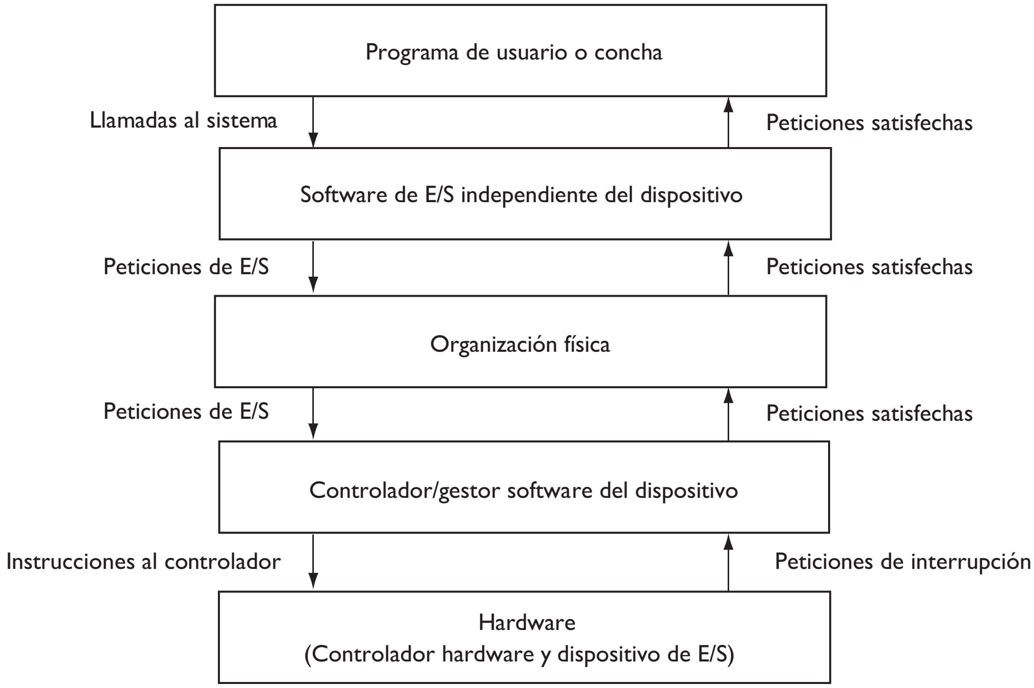


Figura 13.24. *Niveles conceptuales seguidos para la gestión eficiente de las E/S.*

Nivel de procesos de aplicaciones (programa de usuario)

Cuando se dispone de un programa con E/S, antes de su ejecución se enlaza con procedimientos de una biblioteca de E/S. El objetivo básico de estos procedimientos es situar los parámetros de E/S en la zona de memoria adecuada y hacer las llamadas al sistema operativo. Las llamadas al sistema también son generadas por las órdenes del sistema operativo (concha) que involucran a los distintos periféricos.

En el caso del acceso a impresoras compartidas y otros periféricos lentos, así como transferencia de archivos por la red (incluyendo transferencias de correo electrónico) actúa

el sistema de **operación simultánea de periféricos fuera de línea** (**Spool**, *Simultaneous Peripherical Operation Off Line*). Con este sistema cuando, por ejemplo, se desea imprimir un archivo, se realiza una copia del mismo en un área temporal de memoria organizada como un directorio *Spool* de impresión. Concurrentemente con la copia de archivos en el **directorio Spool**, se ejecuta un proceso (genéricamente denominado *demonio*) que es el único que puede leer del directorio *Spool* y que se encarga de ir imprimiendo los archivos de ese directorio y borrándolos cuando concluye la impresión. De esta manera se consigue que:

1) El proceso de impresión se realice virtualmente de forma rápida, ya que desde el punto del vista de la aplicación del usuario la impresión finaliza en el momento que el archivo de impresión se incluye en el directorio *Spool,* independiente del estado actual del periférico.
2) Se gestione adecuadamente la utilización del periférico por diversos usuarios.
3) El procesador se utilice eficientemente ya que el demonio encargado de la impresión es un proceso más que se ejecutará concurrentemente con los demás procesos activos en ese momento.

Puede afirmarse que el módulo *Spool* del sistema operativo hace que los periféricos incluidos en él puedan verse por las aplicaciones como **dispositivos virtuales**.

Software de E/S independiente del dispositivo

En este nivel, presentando una interfaz uniforme al nivel de usuario, se incluyen funciones de E/S comunes o genéricas a todos los dispositivos, tales como abrir, cerrar, leer y escribir. Según se indicó anteriormente en UNIX y Windows NT los dispositivos se utilizan como archivos especiales, por lo que este nivel se corresponde con el **sistema de gestión de archivos** (véase Sección 13.8). Algunos de los cometidos de este nivel son los siguientes:

a) A partir de la denominación simbólica del archivo se localizan las tablas que contienen los atributos del dispositivo (donde se encuentran las direcciones tanto del controlador físico como de la unidad concreta y volumen), y las posiciones físicas donde se encuentran almacenados los distintos bloques del archivo (Tabla de índices y la tabla maestra de ficheros, en UNIX y Windows_NT, respectivamente, véase Sección 13.8.1); en definitiva, se realiza una traducción de nombres simbólicos a direcciones físicas.
b) Protección de dispositivos, controlando los derechos de acceso: *escribir, leer, ejecutar*.
c) Asignación y liberación de dispositivos dedicados, informes de errores, etc.

Organización física

En este nivel ya se tienen en cuenta las cuestiones específicas de cada tipo de periférico (impresora, ratón, teclado, pantalla, disco, etc.). Fundamentalmente tiene los siguientes cometidos:

a) Realizar las funciones de alto nivel específicas del periférico o del tipo de periférico al que se le solicita la operación de entrada o salida; así en el caso de un disco se hace la gestión de estructuras RAID (ver Sección 10.3.2), se agrupa la información al tamaño requerido para formar bloques físicos ya que el nivel de E/S independiente del dispositivo considera a todos los archivos (incluyendo a los de disco) como un conjunto sucesivo de información; también se gestiona la ubicación física de los bloques físicos y de los espacios libres, etc.
b) Utilizar técnicas de almacenamiento intermedio (*buffering*) en la RAM para aumentar el rendimiento; entre estas técnicas se encuentran la **caché de E/S** (*buffer cache*) de UNIX o la **caché de archivos** (*file cache*) de Windows NT; la caché de archivos o de disco es una técnica que consiste en, una vez localizado el cilindro donde se

encuentra el sector al que se quiere acceder, leer todo el cilindro completo (o incluso varios adyacentes), cosa que se puede realizar a la velocidad de rotación del disco. Esta información se almacena en una zona de la memoria principal, de forma que si el programa del usuario ordena el acceso sucesivo a sectores de un mismo cilindro o cilindros adyacentes (cosa muy probable a causa de la localidad espacial de los datos) no se tendrán que consumir tiempos adicionales de latencia para acceder a los distintos sectores; cuando acaban las operaciones con el disco (se cierra el archivo) o hay que desalojar la información previa de la caché, se actualiza el disco físico; esta técnica se gestiona de forma similar a la memoria virtual.

c) Convierte las referencias lógicas a archivos y registros en direcciones físicas, teniendo en cuenta la estructura física del dispositivo de E/S.

Controladores software de los dispositivos hardware

Fundamentalmente este nivel tiene los siguientes cometidos: a) planificación y control, b) generación de las instrucciones concretas para interactuar con el hardware, y c) atención de interrupciones. A continuación se describen brevemente.

a) Planificación y control: se planifican las secuencias de instrucciones de E/S, atendiendo la cola de accesos. Para ilustrar con un ejemplo una situación de planificación, a continuación se considera un caso particular de disco magnético. Los controladores de disco, sobre todo cuando se configuran como **servidores de archivos,** donde se prevé gran cantidad de peticiones sucesivas de acceso desde distintos procesos, deben implementar algoritmos que mejoren las prestaciones, fundamentalmente la velocidad.

Según analizamos en la Sección 10.3.2, el tiempo de lectura o escritura de un sector del disco depende de tres factores: tiempo de búsqueda del cilindro, tiempo de latencia (espera al sector) y velocidad de transferencia. El tiempo que se puede optimizar desde el programa gestor del disco es el tiempo de búsqueda del cilindro.

La forma más fácil de gestionar la búsqueda del cilindro es hacerla directamente según se van acumulando las peticiones de acceso: cuando éstas se acumulan, se almacenan en una memoria FIFO de tal manera que la primera solicitud que llega es la primera que se atiende; es el **algoritmo FCFS** (*First-Come First-Served,* primero en llegar, primero en ser atendido). Otros algoritmos, con los que se consigue un mejor rendimiento del disco, se basan en mantener una tabla con tantos bits como cilindros. Todos los bits inicialmente están a cero, y cuando hay una solicitud de un cilindro se pone a 1 el bit correspondiente; es decir, en un momento dado habrá tantos unos como peticiones de acceso a cilindros pendientes. Una vez realizado el acceso a un cilindro se pone a cero el bit correspondiente. El **algoritmo SSF** (*Shortest Seek First,* el más cercano se atiende primero), va seleccionando los cilindros en función de la distancia más corta a donde se encuentre posicionada la cabeza. Este algoritmo presenta el problema de que puede haber peticiones que se retrasen mucho, ya que constantemente están llegando peticiones; como consecuencia se accederá más rápidamente a las peticiones de los cilindros de las zonas centrales; por ejemplo, en un disco con 80 cilindros se atenderán por término medio con mayor prontitud los cilindros del 40 al 60. Otro algoritmo, denominado **algoritmo SCAN** o **del ascensor**, recorre la tabla de bits en secuencia, de abajo a arriba (de 0 a 79) accediendo a los cilindros con peticiones pendientes (es decir, los que en la tabla tienen un 1) y posteriormente del cilindro superior que haya tenido petición, en dirección al cilindro inferior con petición, y así sucesivamente.

EJEMPLO 13.16

Suponiendo, para simplificar, que un disco contiene tan sólo 22 pistas, y que su peine de cabezas se encuentra inicialmente posicionado en el cilindro 11 y se acumulan sucesivamente las solicitudes de acceso a los siguientes cilindros: 21, 6, 13, 5, 10, los accesos se efectuarían en el siguiente orden (Figura 13.25):

Algoritmo FCFS:

11 → 21 → 6 → 13 → 5 → 10

Algoritmo SSF:

11 → 10 → 13 → 6 → 5 → 21

Algoritmo ascensor:

11 → 10 → 6 → 5 → 13 → 21

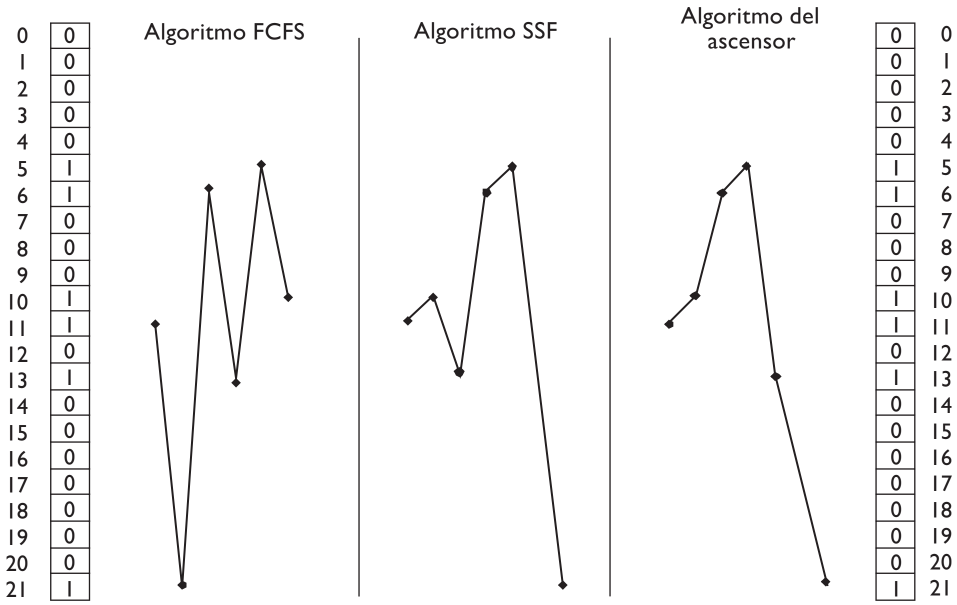


Figura 13.25. *Representación de los movimientos que se producen en la cabeza de una unidad de disco para atender la secuencia de peticiones del Ejemplo 13.8.*

b) **Generación de las instrucciones concretas para interactuar con el hardware:** los dispositivos físicos de los periféricos contienen en general (Sección 9.5.1) puertos de control, donde deben ubicarse las órdenes concretas que se deben realizar (arrancar, parar, recalibrar, buscar, leer, escribir, leer, dar formato, etc., en caso de un disco, por ejemplo), y puertos de estado, donde después de realizada la operación el controlador incluye una palabra binaria que indica el resultado de la operación. También contiene puertos para los datos a escribir o leer del dispositivo de E/S que usualmente se transfieren a una segunda memoria temporal ubicada físicamente en el propio dispositivo. El controlador software del dispositivo debe traducir todas las peticiones (abstractas)

que le llegan de la capa inmediatamente superior a secuencia de operaciones que, en el controlador hardware del periférico concreto:

- Carguen las instrucciones en el puerto de control.
- Lean o escriban del puerto de datos la información a recibir/transmitir al periférico.
- Lean del puerto de estado el estado del periférico.

Como ejemplo considérese el caso de una unidad de disco al que el nivel inmediato superior le presenta una operación de lectura de un sector, el controlador debe comprobar si el disco está arrancado (y si no dar la orden de arrancarlo), posicionar el brazo en el cilindro correspondiente, inicializar el controlador de DMA, transmitir el marco de página de información, etc. En realidad los controladores de periféricos son los únicos programas que deben tener en cuenta las peculiaridades concretas de los dispositivos.

c) **Atención de interrupciones y detección de estado del periférico:** cuando se inicia una operación de E/S, el estado del proceso pasa a bloqueado, esperando a que acabe la operación. En la mayoría de los dispositivos de E/S, las operaciones de lectura y escritura se inician y finalizan por medio de interrupciones. Cuando se produce la interrupción final, se lanza una rutina de este nivel que capta la información del registro de estado del dispositivo hardware correspondiente, y si la operación de E/S se ha realizado mal, se vuelve a intentar o se informa a los niveles superiores para generar el mensaje correspondiente. Si el resultado ha sido correcto, la rutina de interrupción cambia en la tabla de procesos el estado del proceso que ha finalizado la operación de E/S de bloqueado a preparado, para que así pueda continuar su ejecución cuando el distribuidor le dé turno.

Dispositivos hardware

Este nivel está constituido por los *controladores hardware de E/S*, en general (controladores DMA, procesadores de E/S, tarjetas controladoras, etc.) que son los elementos físicos que realmente implementan la operación de E/S. Todos estos elementos contienen diversos puertos de E/S (control, estado, y de datos) que son monitorizados por el nivel inmediato anterior (control software del dispositivo).

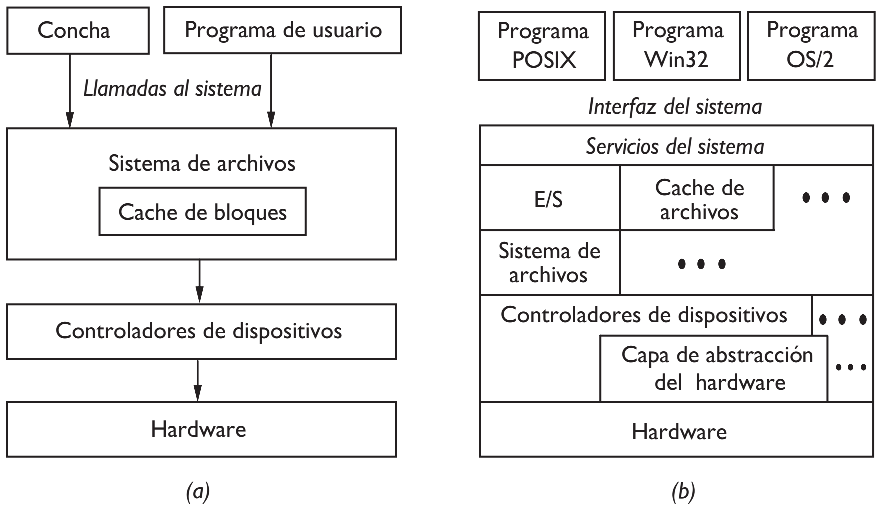


Figura 13.26. *Modelo simplificado para gestión de E/S seguido en los sistemas operativos (a) UNIX, y (b) Windows NT.*

Los principios conceptuales de este modelo son seguidos por la mayoría de los sistemas operativos, aunque en los detalles pueden diferir. En la Figura 13.26 se incluye un esquema que muestra a grandes rasgos la estructura de los sistemas operativos UNIX y Windows NT, en lo que atañe a la gestión de E/S.

13.8 Gestión de archivos. El sistema de archivos

En esta sección vamos a referirnos a la porción del sistema operativo que coordina el uso de los dispositivos de memoria masiva, manteniendo información de los nombres simbólicos de todos los archivos almacenados, del lugar físico donde se encuentran ubicados, de los usuarios que pueden acceder a cada uno de ellos, de la localización de las zonas libres, etc.

Un archivo, dependiendo principalmente del uso o capacidad que vaya a tener, puede estructurarse en un soporte (disco o cinta) de distintas formas. Hay un sistema físico más o menos complejo, y una visión lógica adecuada al usuario. El sistema operativo hace posible utilizar esta última, haciendo de interfaz entre el sistema físico y el usuario. En efecto, desde el punto de vista hardware, para almacenar datos o programas sólo existen direcciones físicas. En un disco toda información se graba o lee físicamente en bloques referenciados por (número de unidad)/(superficie)/(pista)/(sector). El sistema operativo posibilita que el usuario no tenga que utilizar direcciones físicas: podemos actuar sobre un archivo y almacenar o recuperar información sin más que indicar su nombre y utilizar llamadas al sistema o ciertas instrucciones del lenguaje de control del sistema operativo.

Sobre la estructura hardware descrita (referenciada por direcciones físicas) el sistema operativo construye dos abstracciones: la de **archivo** y la de **directorio** (o **carpeta**). El conjunto de módulos del sistema operativo que se encargan de la gestión de archivos y directorios se suele denominar **sistema de archivos**.

13.8.1 Gestión de archivos

El concepto de **archivo** posibilita aislar al usuario de los problemas físicos de almacenamiento. En efecto, cuando el usuario desee referirse a un conjunto de información del mismo tipo como una unidad de almacenamiento, no tiene nada más que crear un archivo dándole el nombre que considere oportuno. Los archivos, que pueden ser tanto de programas como de datos, se conciben como estructuras con las siguientes peculiaridades:

- Deben ser capaces de contener grandes cantidades de información.
- Su información debe permanecer y sobrevivir a los procesos que la generan o utilizan.
- Distintos procesos deben poder acceder concurrentemente a la información de un archivo.

Dependiendo del sistema operativo se pueden hacer unas u otras operaciones con archivos, algunas de las llamadas al sistema más comunes se incluyen en la Tabla 13.8 (por claridad se omiten los parámetros de las llamadas).

Cada archivo tiene asociado un **nombre** (dado arbitrariamente por el usuario siguiendo unas normas preestablecidas, a veces pueden ser varios), **atributos**, **descriptor de seguridad** (especifica los grupos de personas que pueden acceder al fichero y sus privilegios), **dirección** donde se encuentran los datos, y los **datos**. Los atributos pueden incluir cuestiones tales como fecha y hora de creación, fecha y hora de la última actualización, bits de protección (sólo lectura, o lectura y escritura), contraseña de acceso, número de bytes por registro, capacidad del archivo, etc.

Tabla 13.8. *Llamadas al sistema más comunes relacionadas con la gestión de archivos.*

Llamada al sistema	UNIX	Win32 y Windows NT
Crear archivo	open	CreateFile
Borrar archivo	unlink	DeleteFile
Cerrar archivo	close	CloseHandle
Leer datos de un archivo	read	ReadFile
Escribir datos en un archivo	write	WriteFile
Llevar el puntero del archivo a una posición determinada	lseek	SetFilePointer
Obtener las propiedades del archivo	stat	GetFileAtributes
Bloquear una zona del archivo para evitar el acceso simultáneo de otro usuario	fcntl	LockFile
Desbloquear una zona del archivo	fcntl	UnlockFile

Desde el punto de vista del programador o de las aplicaciones, los archivos se organizan en registros lógicos, formados por campos. Los registros lógicos se definen en función de la aplicación o aplicaciones que los vayan a utilizar, no estando, por tanto, su tamaño definido a priori.

Por otra parte, los archivos se almacenan en los dispositivos de memoria masiva en forma de unidades, denominadas registros físicos o bloques. Un **registro físico** es la cantidad mínima de información que puede transferirse en una operación de entrada/salida, y su tamaño viene determinado por el dispositivo físico y el sistema operativo. Así, las últimas versiones de Windows de Microsoft utilizan tamaños de 2 ó 4 KBytes para cada bloque (que denominan **unidad de ubicación** o ***clúster***). En consecuencia, no existe de antemano ninguna relación entre el tamaño de un registro lógico y el de un bloque: un registro lógico por lo general ocupa parte de un bloque, aunque puede ocupar varios de ellos. Se denomina **factor de blocaje** al número de registros lógicos que se pueden almacenar en un registro físico. El sistema operativo se encarga de gestionar el almacenamiento de los registros lógicos. Cuando un programa quiere acceder al contenido de un archivo genera una llamada al sistema operativo y éste lee los registros físicos necesarios, los almacena en una zona de la memoria principal asociada a la entradas/salidas del archivo denominada ***buffer*** y el programa de aplicación accede a ese *buffer* para leer el registro lógico o datos que haya solicitado. De forma similar, cuando una aplicación quiere almacenar información en un archivo, pasa los datos al sistema operativo, que los almacena en el *buffer* y cuando se completa un registro físico lo transfiere al dispositivo de memoria masiva (Figura 13.27).

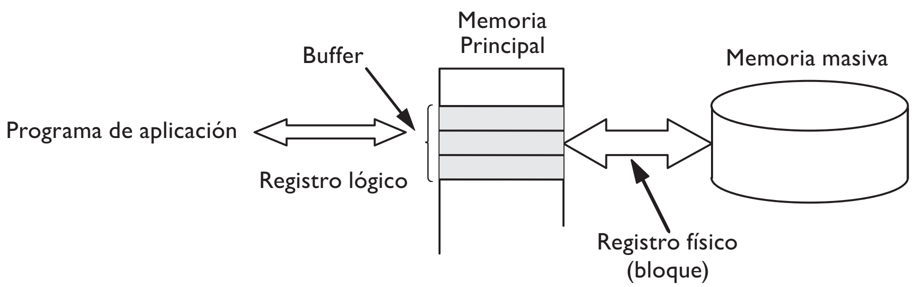


Figura 13.27. *Actuación del sistema operativo en la gestión de almacenamiento y lectura de archivos.*

Para poder llevar a cabo las operaciones de lectura y escritura de un archivo, el sistema operativo tiene que conocer cierta información sobre éste, como su nombre, en qué dispositivo está almacenado, el *buffer* utilizado, etc. Esta información la guarda en una tabla en memoria principal llamada **descriptor de archivos**, y se crea cuando una aplicación le comunica al sistema operativo que quiere acceder a un archivo (**abrir el archivo**), y se elimina cuando la aplicación le dice al sistema operativo que ya no va a necesitar más ese archivo (**cerrar el archivo**).

Cuando damos al sistema operativo la orden de almacenar un archivo (de un programa, de un texto, vídeo o música, por ejemplo) utiliza una de las siguientes formas:

- **Contigua:** los bloques se almacenan consecutivamente uno tras otro. Esta forma es la usual en los CD. Como los archivos no suelen tener una longitud fija a lo largo de su vida, esta forma de grabación presenta serios problemas: no se conoce a priori el tamaño máximo del archivo, con lo que se podrían producir desbordamientos, y si se conociese se producirían **fragmentaciones** (es decir, espacios entre archivos desperdiciados) cuando el archivo no estuviese a su máxima capacidad.
- **Lista encadenada:** los bloques se almacenan en **unidades de ubicación** (*clusters*) no necesariamente consecutivas, encadenándose un bloque con otro por medio de un puntero. Se accede al archivo conociendo la dirección de la unidad de ubicación inicial. Con este sistema no hay fragmentación; pero, sin embargo, presenta el problema de que para acceder a un bloque concreto es necesario acceder secuencialmente (Figura 13.28), desaprovechándose la facilidad de acceso directo a bloque de los discos.

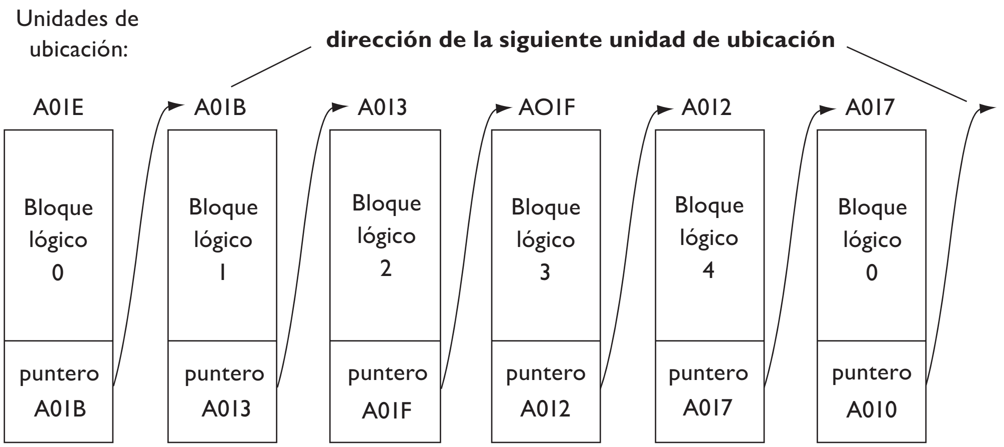


Figura 13.28. *Grabación y localización de las unidades de asignación con una lista encadenada.*

- **Lista de enlaces*:*** cada disco dispone de una tabla con tantos elementos como bloques físicos, la posición de cada elemento se corresponde biunívocamente con cada bloque, y contiene el puntero al lugar donde se encuentra el siguiente bloque del archivo (Figura 13.29). Cuando se abre un archivo, el sistema de archivos carga en la memoria principal la lista de enlaces, pudiéndose obtener rápidamente las direcciones de los bloques consecutivos del archivo. Presenta el inconveniente de que si el disco es muy grande, la lista de enlaces ocupa una capacidad excesiva en memoria principal.

EJEMPLO 13.17

El MS-DOS y Windows utiliza para grabación de la información una lista de enlaces denominada **FAT** *(Tabla de localización de archivos). Se utilizan dos alternativas FAT16 y FAT32:*

- *La versión FAT16 sirve para discos de hasta 2 Gbytes (unidades de asignación de 32 KB), cada dirección de bloque se da con 2 bytes. Como cada elemento de la FAT corresponde a un bloque, se puede grabar en él información alternativa al puntero; por ejemplo indicar si el bloque está deteriorado (H'FFF7, en MS-DOS), si está libre (H'0000) y por tanto disponible para su uso, o si es el último del archivo (H'FFFF).*
- *En la versión FAT32 (últimas versiones de Windows 95 –OSR2– y Windows 98), sirve para discos de hasta 1 TB/2 TB, siendo, sin embargo, las unidades de asignación menores (2 KB/4 KB), lo que implica un mejor aprovechamiento del disco. Cada elemento de la FAT tiene 32 bits, de los que 29 especifican el número de la unidad de asignación.*

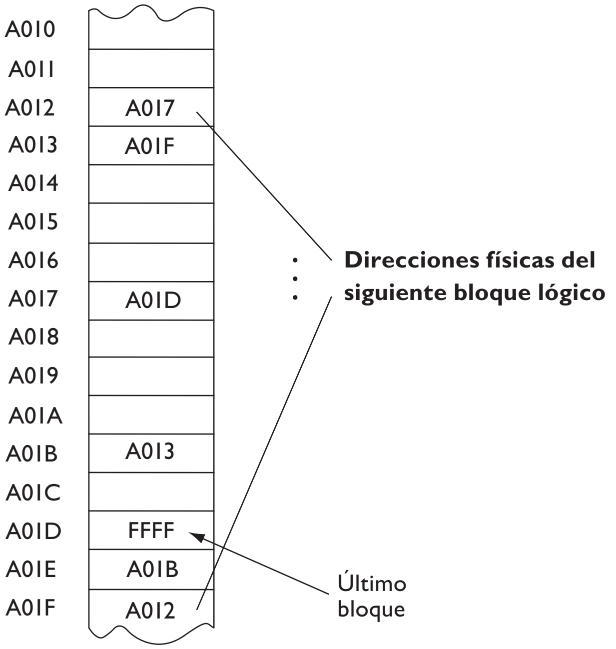


Figura 13.29. *Grabación y localización de las unidades de asignación con una lista de enlaces.*

- **Fichero de índices o i-nodos:** corresponde a la forma de gestionar los archivos por el sistema operativo UNIX. Los datos que se dan a continuación corresponden a UNIX versión 7. Cada archivo tiene asociado un **nudo de información** o **i-nodo** que es una pequeña tabla de tamaño fijo (64 bytes) conteniendo los atributos del archivo y 13 direcciones de 3 bytes. Las 10 primeras direcciones indican directamente las posiciones de las 10 primeras unidades de asignación donde se encuentran los datos del archivo, mientras que las 3 últimas indican las siguientes direcciones de los bloques del archivo,

pero de forma indirecta a través de bloques de índices: con simple indirección, doble indirección y triple indirección, respectivamente, según se indica en la Figura 13.30. Cada bloque de índices puede tener, por ejemplo, 256 direcciones a otros bloques de índices o a unidades de asignación de datos del archivo. En UNIX Sistema V el tamaño de las unidades de asignación es de 1 Kbyte, con lo que, con el esquema anterior, la capacidad de un archivo puede llegar a 16 Gbytes.

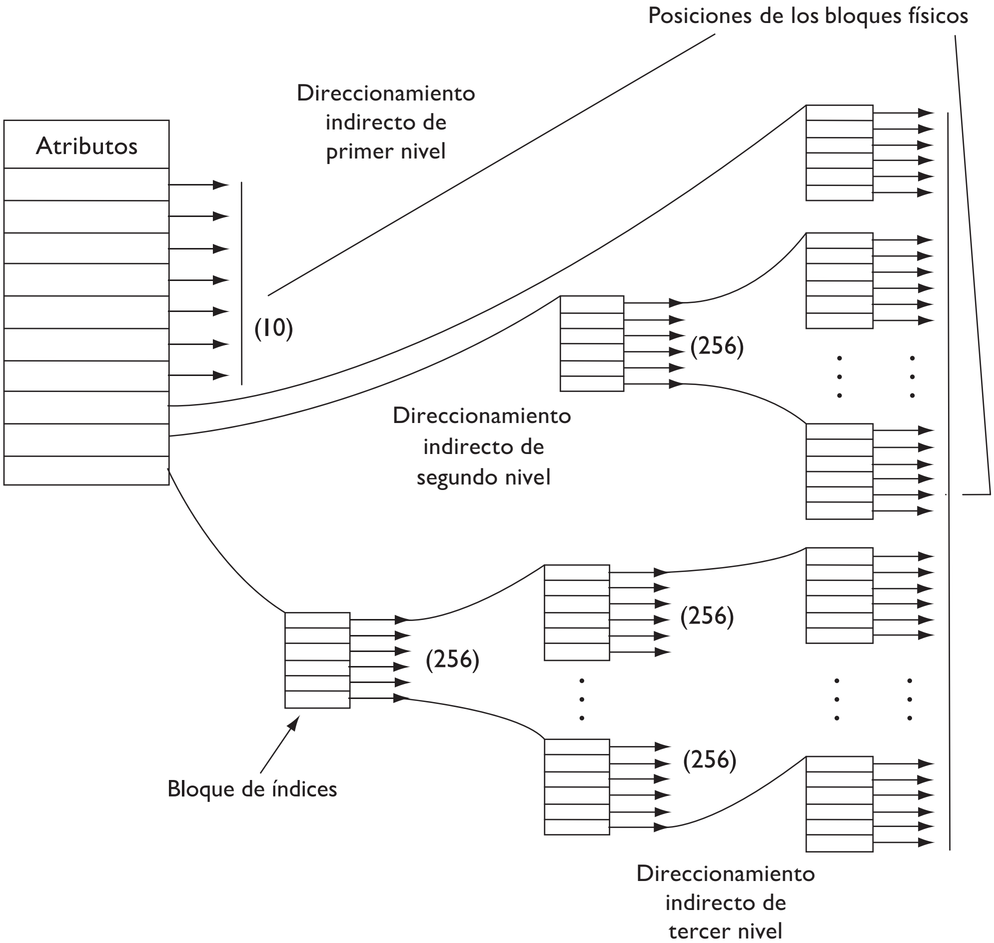


Figura 13.30. *Grabación y localización de las unidades de asignación con un fichero de índices.*

EJEMPLO 13.18

En la versión 4.2BSD de UNIX: los nodos de índices tienen 12 punteros directos, el tamaño de las unidades de asignación (bloques) es de 4 KB, cada puntero ocupa 4 bytes. El tamaño máximo de los archivos es de 2 GB.

Windows NT admite varios sistemas de archivos, como FAT16, FAT32 y OS/2, siendo original en él el **sistema de archivos de nueva tecnología (NTFS**, *NT File System*). Las unidades de almacenamiento que se consideran en NTFS son:

- Sector: unidad de almacenamiento más pequeña (usualmente de 250 Bytes).
- Unidad de ubicación: uno o más sectores contiguos, en la misma pista (de 512 Bytes a 64 Kbytes).
- Volumen: porción lógica de un disco compuesta de una o más unidades de ubicación (de 1 a 128 sectores/volumen).

Con Windows NT se pueden tener ficheros de hasta 2^{64} Bytes. Las unidades de ubicación, dentro de un determinado volumen, se especifican con 64 bits. Cada volumen contiene:

- Región de arranque de la partición (puede ocupar hasta 16 sectores), con información para arranque del volumen, y sobre el propio volumen (trazado y estructuras contenidas en el volumen).
- Tabla maestra de archivos (MFT, *Master File Table*), con información acerca de los archivos y carpetas dentro del volumen, e incluso los propios datos del archivo.
- Archivos del sistema: entre los que se encuentran archivos redundantes ("espejos") para recuperación de fallos, lista de transacciones realizadas, tabla de definición de tipos de atributos admitidos por el volumen, mapa que especifica las unidades de asignación que están ocupadas y libres.
- Zona de expansión para datos de archivos.

El elemento fundamental del volumen es la tabla maestra de archivos que suele tener un elemento (registro) de longitud variable por archivo o carpeta del volumen. Suele contener, entre otras, lo siguiente:

- *Información estándar* que consiste en los atributos del sistema.
- *Nombre(s) del fichero,* que puede ser de hasta 255 caracteres Unicode. Admite varias denominaciones simultáneas para un mismo fichero, en particular las MS-DOS.
- *Seguridad,* puntero a la zona referente al archivo, que se encuentra en un archivo centralizado que contiene toda la información de seguridad.
- *Datos,* si el archivo es pequeño, su contenido se encuentra en el propio registro de la tabla (**archivos inmediatos**), si no contiene punteros a las unidades de asignación de la zona de expansión donde se encuentran los datos sucesivos o, usualmente, las direcciones iniciales de las unidades de asignación donde realmente se encuentran los datos, junto con la longitud de los datos del archivo.

13.8.2 Gestión de carpetas o directorios

La segunda abstracción que utiliza el sistema operativo para gestionar volúmenes de datos es la de **directorio** o **carpeta**. Los directorios o carpetas son conjuntos de archivos agrupados siguiendo algún criterio arbitrariamente elegido: directorio del usuario que lo crea, directorio de cartas, directorio de facturas, etc. La estructura global del sistema de archivos suele organizarse en forma de árbol en el que los nodos interiores son directorios o archivos y los nodos exteriores son archivos. De un directorio pueden depender, Figura 13.31, archivos u otros directorios (también denominados **subdirectorios**).

En la Tabla 13.9 se incluyen las llamadas al sistema más comunes relacionadas con la gestión de directorios o carpetas.

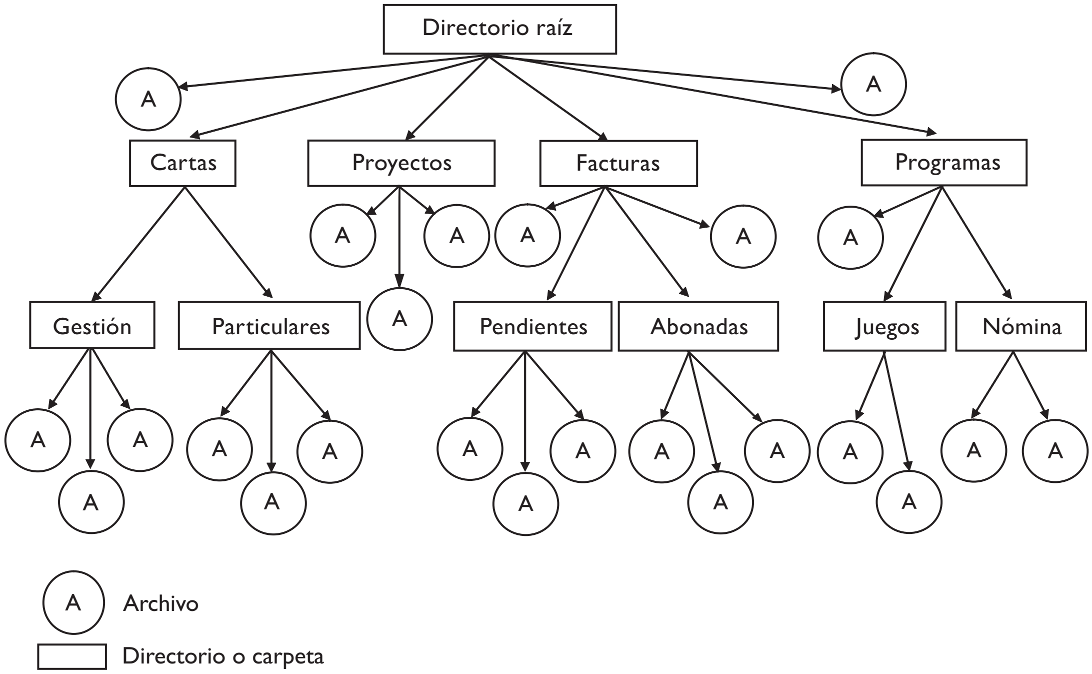


Figura 13.31. *Ejemplo de organización de archivos y directorios en forma de árbol.*

Tabla 13.9. *Llamadas al sistema más comunes relacionadas con la gestión de directorios.*

Llamada al sistema	UNIX	Win32 y Windows NT
Crear un nuevo directorio	mkdir	CreateDirectory
Borrar un directorio vacío	rmdir	RemoveDirectory
Abrir un directorio para lectura (posicionarse en el primer elemento)	opendir	FindFirstFile
Leer el siguiente elemento del directorio	readdir	FindNextFile
Eliminar un archivo de un directorio	unlink	
Llevar un archivo de un directorio a otro		MoveFile
Cambiar el directorio actual de trabajo	chdir	SetCurrentDirectory

Un directorio se puede gestionar con una tabla-índice que contiene un elemento por cada archivo o directorio dependiente de él. Cada elemento está formado por el nombre del archivo dado por el usuario e información adicional, puede estar constituida por los atributos del archivo y el bloque donde comienza el archivo, caso del MS-DOS (recuérdese que a partir de conocer el bloque inicial del archivo con la FAT se pueden conocer el lugar donde se encuentran los bloques sucesivos). También la información adicional puede ser un puntero a otra estructura con información sobre el archivo; este es el caso del UNIX, en el que el puntero sencillamente es la dirección del i-nodo del archivo, que, como hemos visto anteriormente, contiene tanto los atributos del archivo como la tabla de posiciones. En el sistema NTFS la información sobre las carpetas se incluye en registros de la MFT.

Cuando se abre un archivo, el sistema operativo utiliza la **ruta** (*path-name*), que contiene una lista de todos los directorios atravesados desde el directorio raíz al archivo en cuestión (al que identifica por su nombre), y extrae a partir de la información del elemento la tabla de direcciones en disco y la ubica en memoria principal. Con ayuda de esta tabla rápidamente pueden realizarse todas las referencias al archivo.

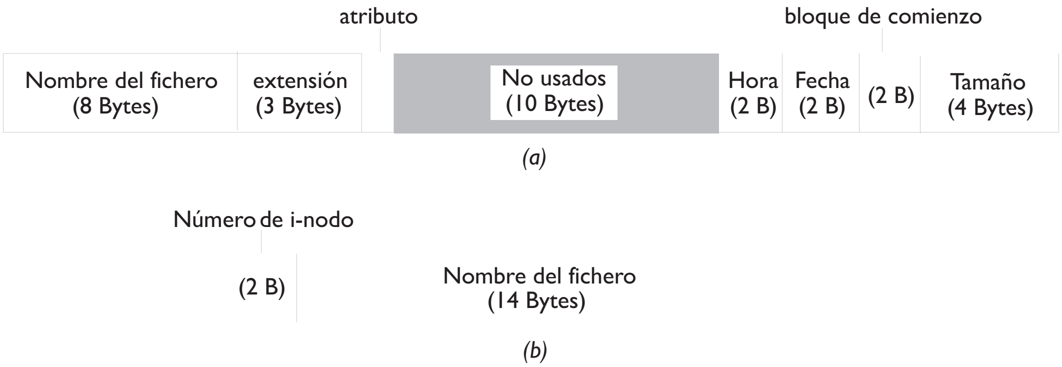


Figura 13.32. *Elemento de una tabla-índice: (a) de un directorio MS-DOS; (b) de un directorio UNIX.*

A título de ejemplo, en MS-DOS la información correspondiente a cada archivo que contenía el directorio era (Figura 13.32):

- *Nombre del archivo:* 8 bytes, y se almacena en código ASCII
- *Extensión:* 3 bytes, código ASCII
- *Atributos:* 1 byte,
  - H'1 archivo sólo lectura
  - H'2 archivo oculto
  - H'4 archivo creado por el sistema
  - H'8 etiqueta de volumen
  - H'10 subdirectorio
  - H'20 archivo
  - H'40 no utilizado
  - H'80 no utilizado

  El atributo total se obtiene sumando los valores anteriores.
- *Tiempo:* 2 bytes, indicando la hora en que se efectuó la última modificación
  tiempo = hora · 2048 + minutos · 32 + segundos/2
  (Ejemplo, 11:32:10, se almacena como 23557)
- *Fecha:* 2 bytes, indicando la fecha en que se efectúo el último cambio.
  fecha = 512 · (año-1980) + 32 · mes + día
  (Ejemplo, 12/12/1984 se almacena como 2444)
- *Unidad de ubicación inicial:*
  2 bytes, indicando el clúster donde comienza el archivo, este valor sirve también de puntero de entrada para la FAT.
- *Tamaño del archivo:*
  4 bytes, codificado como un entero sin signo.

En UNIX los elementos del directorio son muy sencillos. En la versión 7 dichos elementos contienen sólo el número de i-nodo (2 bytes) y el nombre del archivo (14 bytes) (Figura 13.32b), ya que los atributos del archivo y direcciones para acceso a los datos del mismo se encuentran en el i-nodo. En UNIX, además del directorio raíz, se pueden definir subdirectorios *bin*, para programas, *dev,* para archivos de E/S, *lib,* para bibliotecas, y *usr,* directorios de usuario. Los elementos de directorio en UNIX 4.2BSD son más complejos, ya que son de longitud variable, por permitir que los nombres de archivo puedan tener un número arbitrario de caracteres menor de 256.

13.9 Conclusión

Resulta extremadamente difícil utilizar un computador a partir del conocimiento de su lenguaje máquina; por lo que se dispone de programas de control (sistema operativo) y utilidades generales que facilitan notablemente su uso y hacen aumentar el rendimiento de sus distintas unidades. Se puede considerar así al computador como una máquina virtual en los niveles de **máquina operativa** (hardware junto con el sistema operativo) y **máquina simbólica** (máquina operativa junto a software de utilidades y de aplicaciones).

Se ha hecho especial hincapié en la descripción de las funciones y elementos que configuran un sistema operativo. En el capítulo siguiente (Capítulo 14) se describirán con cierto detalle los aspectos relacionados con los lenguajes de programación, pieza fundamental para la obtención de una **máquina simbólica**.

Los elementos del software de un computador que se incluyen en este capítulo se han presentado de forma descriptiva, ya que únicamente se pretende que el lector comprenda cuáles son los objetivos y funcionamiento básico de cada uno de ellos.

EJERCICIOS

13.1 Un procesador funciona a una frecuencia de F = 1,2 GHz, y, por término medio, cada instrucción consume 10 ciclos de reloj. El computador dispone de una unidad de disco que gira a una velocidad de 10.800 rpm, el tiempo de búsqueda de pista es de 5 ms, y el ancho de banda entre el disco y la memoria es de 20 MB/s. Se tiene un proceso que, en su inicio, carga desde disco un archivo de 1 MB, después lo actualiza, y al final lo salva en el disco. El proceso, en total, tiene 834.000 instrucciones.

a) Suponiendo un sistema de monoprogramación, estimar el tiempo de ejecución y el coeficiente de respuesta para el proceso.

b) ¿Cuántas instrucciones podría haber ejecutado el procesador mientras el proceso permanece bloqueado?

c) Suponiendo que el sistema operativo fuese de multiprogramación con un quantum de T = 40 ms, ¿durante cuántos quanta estaría bloqueado el proceso?

13.2 En un instante dado entran en una cola serie de un sistema de multiprogramación tres programas, con la estimación de tiempos de ejecución que se dan a continuación:

P0(15 minutos); P1 (9 minutos); P2 (10 minutos); P3 (2 minutos)

¿Con cuál de las siguientes técnicas de planificación obtendría P1 mejor tiempo de respuesta: FCFS *(First Come First Seved),* SPN *(Shortest Process Next)* o SRT *(Shortest Remaining Time)*?

13.3 Un computador dispone de un espacio de memoria de 8 MBytes para los programas de los usuarios. Suponga que se tiene una cola con tres programas (P1, P2, P3), cuyas principales características se dan en la tabla: tiempo de procesamiento (s), tiempo de E/S (w1), instante en que, una vez iniciado el proceso correspondiente, se tiene la operación de E/S (suponer que es única), y la capacidad de memoria que ocupa en total el proceso. Los

tiempos se dan en milisegundos. Obtener el rendimiento, tiempo de ejecución de cada proceso, coeficiente de respuesta medio, tasa de ocupación del procesador, y tasa de ocupación de memoria, para los siguientes casos:

a) Monoprogramación.

b) Multiprogramación no apropiativa.

c) Multiprogramación apropiativa, turno rotatorio, con quantum T = 20 ms.

	P1	**P2**	**P3**
Tiempo de procesamiento (s)	80	50	100
Tiempo de E/S (w_l)	60	80	50
Instante de E/S	10	40	15
$C_{memoria}$ (Mbytes)	2	5	1

13.4 En una cola serie se encuentran cinco programas, con la estimación de tiempos de ejecución que se indica a continuación:

P0 (10 minutos); P1 (6 minutos); P2 (4 minutos); P3 (4 minutos); P4 (4 minutos)

Suponiendo que los programas anteriores entran en el computador en el mismo instante, comprobar que el tiempo medio de retorno para los cinco procesos es menor si se ejecutan primero los de menor duración y después los de mayor duración.

13.5 Suponga el siguiente conjunto de procesos:

Nombre del proceso	**Instante de llegada**	**Tiempo de procesamiento**	**Prioridad (5 la mayor)**
P0	0	2	2
P1	1	9	3
P2	2	1	2
P3	2	9	4

Considere los siguientes algoritmos de planificación:

1. Primero en llegar, primero en servir (FCFS).
2. Turno rotatorio (RR), con quanta de T =1.
3. Turno rotatorio (RR) con quanta de T = 4.
4. Primero el proceso más corto (SPN).
5. Menor tiempo restante (SRT) apropiativo (T=2).
6. Planificación por prioridad (considere fija la prioridad indicada en la última columna, y suponga que siempre que llegue un proceso más prioritario que el activo se interrumpe éste para dar paso a aquél).

Para todos los casos anteriores:

a) Hacer un diagrama de tiempos.

b) Obtener los coeficientes de respuestas medio, y el número de veces que tiene que intervenir el SO (para tomar una decisión de planificación o/y realizar un cambio de contexto).

c) Obtener la tasa de utilización del procesador.

d) ¿Qué conclusiones pueden obtenerse de los resultados obtenidos?

13.6 Si la tabla de páginas de un proceso de 15 Kpalabras es la que se indica a continuación, y suponiendo que los bloques de memoria son de 4 Kpalabras, indicar las direcciones físicas de memoria que ocuparán las instrucciones cuyas direcciones virtuales son las siguientes:

a) 0000

b) 3A10

c) 2FFF

Pág	**Bloque nº**
0	5
1	8
2	7
3	6

13.7 En un determinado sistema operativo que utiliza segmentación, la dirección virtual de una instrucción es CD7F A7354, suponiendo que el direccionamiento se hace por palabras de 32 bits, ¿cuál es la capacidad máxima prevista para los programas?

13.8 Un sistema operativo con memoria segmentada utiliza tablas de segmentos de 256 elementos. Cada elemento contiene la capacidad del segmento (16 bits) y la dirección base correspondiente. Sabiendo que, para un determinado programa, el elemento que ocupa el decimotercer lugar de la tabla de segmentos contiene el valor A357 B735 0000, ¿qué dirección o información generaría la unidad de gestión de memoria (MMU) para la dirección lógica B735F?

13.9 Si la tabla de páginas de un determinado proceso es:

0	**5**
1	**A**
2	**B**
3	**7**

¿Cuál es la dirección física de la dirección virtual 2FAC?

13.10 Si la tabla de páginas de un determinado proceso es:

4
A
D
F
3

¿A qué dirección virtual corresponde la dirección física 3FCA2?

13.11 En un sistema de paginación, la capacidad de la página es *Cp* y un programa requiere *Np* páginas. Suponga que por término medio la última página de cada programa está ocupada al 50%. Cada página necesita *p* posiciones en la tabla de marcos de página.

a) Obtener una ecuación que proporcione la cantidad total de memoria requerida para el conjunto del programa y su tabla de páginas.

b) Encontrar analíticamente el tamaño óptimo de página para requerir una capacidad mínima de memoria.

c) Determinar el tamaño de página adecuado para almacenar un programa de 1 MByte suponiendo que se utilizan 4 bytes para almacenar cada elemento de la tabla de páginas.

13.12 Un computador organizado en palabras de 32 bits dispone de un procesador con un tiempo de ciclo de 600 ns y de una unidad de disco que gira a 3.600 rpm, y realiza las transferencias de información a 1,2 MBytes/s. El sistema operativo es de memoria virtual, con páginas de 1 Kpalabra. Analizada la naturaleza de los programas que en ella se procesan, durante un largo periodo de tiempo, se observa que:

a) El 99,2% de las instrucciones que se ejecutan acceden a la página en curso de ejecución.

b) De las instrucciones que acceden a otra página, el 80% de las veces acceden a páginas ya en memoria.

c) Cuando se necesita una nueva página, la página a reemplazar el 1,5% de las veces ha sido modificada.

Hacer, para el sistema anteriormente descrito, una estimación del incremento medio del tiempo de ciclo, a consecuencia de los transvases de páginas asociados a la memoria virtual.

13.13 Un sistema con memoria virtual paginada utiliza 32 bits para las direcciones virtuales de los programas, y cada uno de éstos puede llegar a tener hasta 4 Kpáginas. Por otra parte, la memoria principal tiene una capacidad de 1 Gp. Si como máximo el sistema operativo mantiene 2 páginas de cada proceso en la memoria principal, ¿cuál es el número mínimo de procesos a partir del cual es necesario hacer intercambios *(swapping)* entre memoria principal y disco?

13.14 Suponiendo un computador de longitud de palabra de 32 bits, con una memoria principal de 256 MB y que los procesos que ejecuta ocupan como máximo 64 Kpalabras, ¿cuál es el número mínimo de procesos a partir del cual es necesario hacer intercambios *(swapping)* entre memoria principal y disco?

13.15 Un computador en el que se direccionan palabras de 32 bits es gestionado por un SO con paginación, dándose las direcciones virtuales con 32 bits y pudiendo llegar a tener un proceso hasta 4 Kpáginas. El SO utiliza un disco con un tiempo medio de acceso de 10 ms y con interfaz SCSI-III (16 bits, 40 MB/s). En un momento dado se produce un fallo de página de forma tal que el bit de cambio de la página a desalojar indica que ha sido modificada. Estimar el tiempo que se tardará en efectuar el cambio de página.

13.16 En un sistema operativo con memoria virtual paginada, para especificar las direcciones se yuxtaponen 4 cifras hexadecimales: 1 para referenciar la página y 3 para indicar el desplazamiento dentro de la página.

a) Calcular el tamaño máximo de los programas.

b) Suponiendo que la memoria está inicialmente totalmente vacía, que es de 16 Kpalabras y que se referencian sucesivamente las siguientes direcciones:

B7A0, C328, B547, 3466, C5A7, 7F42, AC3A, B567, 973C

¿Qué referencias causan fallo de página?

Suponiendo que la última página referenciada (973C) ha sido almacenada en el marco de página 1, indicar la dirección física a la que se accede.

13.17 La memoria de un computador es gestionada por el sistema operativo en la forma de memoria virtual con paginación. En un momento dado se encuentran ejecutándose concurrentemente los procesos que se indican

en la siguiente tabla de marcos de página (se indica Proceso/Página):

P0/01, P0/00, P3/02, P2/A6, P4/C5, P1/00, P1/01, P3/00, *libre*, P2/A5

P1/02, P3/01, P2/A4, P2/A3, P5/01, P3/03, P1/03, P4/C4, P2/A1, P4/C6

13.18 En una computadora la memoria principal se estructura en páginas. La tabla de páginas de un proceso es la que se indica en la siguiente tabla:

Página	Marco
0	A6
1	E4
2	F3
3	C8

a) Obtener la dirección virtual de un dato que se encuentra en la posición de memoria F37C44.

b) Obtener la dirección física de memoria que ocuparía la instrucción cuya dirección virtual es: 35B54.

c) ¿Si el tamaño de palabra es de 32 bits, cuál es la capacidad máxima posible de la memoria?

13.19 Un sistema operativo gestiona la memoria principal de un computador organizado en palabras de 32 bits utilizando la técnica de memoria virtual bajo la demanda de páginas de segmentos. El sistema operativo reserva los marcos de página que se indican en la tabla para un determinado proceso PP. El reemplazo de páginas se realiza localmente (es decir, cuando se produce un fallo de página en el proceso PP, la página nueva se introduce en uno de los 4 marcos de página indicados en tabla). La política de reemplazo de páginas es FIFO, y el sistema operativo actúa lo más eficientemente posible para reducir al máximo las transferencias de páginas entre memoria principal y disco.

a) Si se hace referencia ordenadamente a las siguientes direcciones de programa: 51324837, 742B3248, 742C7CD4, 742C3728 Determinar las direcciones físicas (en hexadecimal) a las que se accede en cada caso.

b) Indicar los tamaños máximos de los programas, de los segmentos, y de la memoria principal.

c) Suponiendo que la unidad de disco donde se encuentra el proceso virtual gira a una velocidad de 7.200 rpm, el tiempo de búsqueda de pista es de 5 ms, y que el ancho de banda entre el disco y la memoria es de 20 MB/s, estimar el tiempo necesario para gestionar cada una de las 4 referencias.

Tabla de marcos de página

Marco	Contenido (proceso/segmento/página)	Estado	Instante de carga	Cambio (modificación pág.)
D1	PP,A,A13	0	25	1
D2	PP,3,017	0	25	0
D3	PP,5,137	0	127	0
D4	*(libre)*	1	0	0

13.20 Un computador con un sistema operativo que gestiona memoria virtual con paginación, en un determinado momento tiene la siguiente tabla de marcos de página.

Proceso	Página
SO	01
SO	A3
P1	B4
P3	C3
P2	F3
P3	05
SO	04
	(Libre)
P1	B1
P5	C7
P1	D1
P4	A3
P4	04
P2	A2
P1	D5
P5	CC

a) ¿A qué procesos pertenecen y cuáles son las direcciones lógicas correspondientes a las siguientes direcciones físicas:

B427; 056B; D118?

b) Cuál es la capacidad de memoria principal, la capacidad máxima de los programas y el tamaño de página.

c) Calcular la dirección física a la que se accede cuando se hace referencia a la dirección A312C del proceso P2.

13.21 Un sistema operativo gestiona la memoria principal de un computador organizado en palabras de 32 bits utiliza la técnica de memoria virtual bajo demanda de páginas. Es capaz de gestionar hasta 1 GB de memoria virtual. La memoria principal tiene 4 MB organizados en 16 marcos de página y está inicialmente vacía.

a) Indicar en la segunda columna de la tabla cuándo se producen fallos de página al hacer referencia a las direcciones que se indican en la primera columna

Dirección	Fallo de página	Marco de página	Dirección física
0014569 AF01182 AB01100 0101100 AF01182			
0104400 00AB000 00B0010 0010010			

b) Indicar, en la cuarta columna de la tabla, la dirección física a la que se accede en cada referencia, suponiendo que la memoria principal se va llenando ocupando primero el marco de página de menor dirección que se encuentre libre.

13.22 Suponga un computador cuya capacidad máxima de memoria es de 16 Mpalabras y que un proceso tiene asignados sólo tres marcos de página: 19, 1A y 1B, e inicialmente están libres. Suponga que se hacen referencia a las direcciones virtuales que se indican en la primera columna de la tabla.

a) Indicar el tamaño de las páginas y el tamaño máximo de los programas.

b) Obtener (y escribir en la tabla) las direcciones físicas a las que se accede, haciendo uso de las siguientes políticas de reemplazo:
1) LRU (usada hace más tiempo).
2) FIFO (primera en entrar primera en salir).
3) Reloj (NRU).

Cadena de referencias	Direcciones físicas (en hexadecimal)		
	LRU	FIFO	NRU
A32C754 B43A65C A3275F2 5414576			
7B58798 A3279DC 474678C 7B556FD			
B43667C A32776A 7B54567 A3267CD			

c) Indicar (como resultado del ejercicio) cuál de las políticas es más eficiente, y compararlas desde el punto de vista de recursos que necesitan y el tiempo añadido que necesita el sistema operativo para gestionarlas.

13.23 Suponiendo que una unidad de disco magnético contiene sólo 32 pistas, sus cabezas se encuentran situadas inicialmente en la 13, y que se encuentran acumuladas las siguientes peticiones de acceso (que han ido llegando en el orden indicado): 27, 1, 15, 0, 13, 29, 17;

a) ¿cuál es el número total de pistas que tendrían que atravesar radialmente las cabezas de lectura/escritura con el algoritmo SSF (*Shortest Seek First*)?

b) Si la densidad radial del disco es de 2.490 pistas/pulgada y la velocidad media de desplazamiento del brazo de la cabeza lectora es de 317,5 cm/s, hacer una estimación del tiempo que se tardaría en realizar los accesos a las pistas indicadas.

13.24 Según se ha indicado en el Ejemplo 13.18, en la versión 4.2BSD de UNIX: los nodos de índices tienen 12 punteros directos, el tamaño de las unidades de asignación (bloques) es de 4 KB, cada puntero ocupa 4 bytes. Sabiendo que el tamaño máximo de los archivos reales es de 2 GB:

a) ¿Cuál es la capacidad máxima de los archivos que se pueden direccionar directamente?

b) ¿Cuántos niveles de direccionamiento sería necesario utilizar?
c) ¿Cuáles son las ventajas e inconvenientes de utilizar distintos niveles para el direccionamiento de las unidades de asignación de los datos?

13.25 Comparar la forma de almacenar archivos de 1 Kbyte, 10 Kbytes, 100 Kbytes, 1 Mbyte, 10 Mbytes y 100 Mbytes en Windows 95 (primera versión), Windows'98 y UNIX. Indicar las ventajas e inconvenientes de los distintos sistemas.

13.26 ¿Hasta qué año sería válido el sistema de codificación de fecha de última actualización de un fichero MS-DOS?.

BIBLIOGRAFÍA

[Bro03] Brookshear, J. R.: *Computer Science*; 7ª Edc. Addison Wesley, 2003.

[Car01] Carretero, J.; García, F.; Miguel, P. de, Pérez, F: *Sistemas Operativos*, McGraw-Hill, 2001.

[Dav77] Davis, W. S.: *Operating System: A Systematic View;* Addison-Wesley, 1977.

[Dei94] Deitel, H. M.: *An Introduction to Operating System;* 3ª Edc. Addison-Wesley, 1994.

[Fer98] Fernández, G.: *Conceptos básicos de arquitectura y sistemas operativos. Curso de Ordenadores,* 3ª Ed. Sistemas y Servicios de Comunicación, 1998.

[For03] Forouzan, B. A.: *Foundations of Computer Science,* Thomson, 2003.

[Mil94] Milenkovic, M.: *Sistemas Operativos, conceptos y diseño,* 2ª Edc. McGraw-Hill, 1994.

[Nor03] Norton, P.: *Introduction to Computers,* 5th Ed. McGraw-Hill, 2003.

[Pri05] Prieto, A.; Prieto, B.: *Conceptos de Informática,* Serie Shaum, McGraw-Hill, 2005.

[Sil06] Silberschatz, A.; Galvin, P. B.; Gagne, G.: *Fundamentos de sistemas operativos,* 7ª Edc., McGraw-Hill, 2006.

[Sta05] Stallings, W.: *Sistemas Operativos,* 5ª Edc., Prentice Hall, 2005.

[Tac00] Tackett Jr, J.; Burnett, S.: *Linux,* 4ª Edc., Prentice Hall, 2000.

[Tan01] Tanembaum, A. S.: *Modern Operating Systems.* Prentice Hall, 2º Edc.2001.

[Tan98] Tanenbaum, A. S.; Woodehull, A. S.: *Sistemas operativos. Diseño e implementación,* 2ª Edc. Prentice Hall, 1998.

[Tan00] Tanenbaum, A. S.: *Organización de computadoras. Un enfoque estructurado,* 4ª Edc., Pearson, 2000.

CAPÍTULO 14

Lenguajes de programación. Traductores

Como se señaló en el Capítulo 1 para que un computador funcione es necesario utilizar programas. El programa indica al computador qué tiene que hacer, y ésta únicamente realiza aquellas operaciones que aquél incluye.

Un programa y sus sentencias se construyen o redactan con unos símbolos, y de acuerdo con unas reglas, que constituyen la gramática del **lenguaje de programación**.

Este capítulo se dedica al estudio de los lenguajes de programación, abordando sus elementos, la forma en que se describe su sintaxis, y los fundamentos del proceso de traducción.

Se describen los distintos tipos de lenguajes de programación, y se justifica su existencia; para ello examinaremos las limitaciones de los lenguajes máquina y cómo superar éstas con **traductores**, que permiten utilizar lenguajes de alto nivel. Existe una gran cantidad de lenguajes de alto nivel, por lo que se efectúa una clasificación de los mismos en función de sus distintos campos de aplicación.

14.1 Lenguajes de programación

Un lenguaje de programación es un conjunto de símbolos y de reglas para combinarlos, que se usan para expresar algoritmos. Los lenguajes de programación, al igual que los lenguajes que usamos para comunicarnos, poseen un **léxico** (vocabulario o conjunto de símbolos permitidos), una sintaxis, que indica cómo realizar construcciones del lenguaje, y una semántica, que determina el significado de cada construcción correcta.

Los lenguajes de programación, o lenguajes de alto nivel, están específicamente diseñados para programar computadores. Sus características fundamentales son:

- Son *independientes de la arquitectura física del computador*. Por tanto, no obligan al programador a conocer los detalles del computador que utiliza, y permiten utilizar los mismos programas en computadores diferentes, con distinto lenguaje máquina (*portabilidad*). Existen normalizaciones de lenguajes realizadas por organismos de estandarización (ANSI e ISO, principalmente), las cuales tratan de garantizar la portabilidad de los programas.
- Normalmente, *una sentencia en un lenguaje de alto nivel da lugar, al ser traducida a varias instrucciones en lenguaje máquina*. La Figura 14.1 muestra las instrucciones

ensamblador y máquina a que da lugar la traducción de una instrucción FORTRAN (en un computador Data General Eclipse S-250).

- Utilizan *notaciones cercanas a las habituales* en el ámbito en que se usan. Con estos lenguajes las operaciones se expresan con sentencias o frases muy parecidas al lenguaje matemático o al lenguaje natural. Por los motivos antes aducidos, en los lenguajes de programación se utilizan por lo general palabras o términos en inglés.

Lenguaje de alto nivel	Lenguaje ensamblador	Lenguaje máquina
A = B + C	LDA 0,4,3 LDA 2,3,3 ADD 2,0 STA 0,5,3	021404 031403 143000 041405

Figura 14.1. *Ejemplo de como una sentencia de un lenguaje de alto nivel da lugar a varias instrucciones ensamblador y máquina (sentencia FORTRAN traducida para una DG S-250).*

La utilización de conceptos habituales suele implicar, las siguientes cualidades:

a) Las instrucciones se expresan por medio de *texto,* conteniendo caracteres alfanuméricos, y caracteres especiales (+, =, /, etc.).

b) Se puede asignar un *nombre simbólico* a determinados componentes del programa, para facilitar su comprensión por las personas. En los lenguajes imperativos, el programador puede *definir las variables* que desee, dándoles los nombres que considere oportuno (DEBE, HABER, TOTAL, MEDIA, etc.), siendo las reglas para denominación de las mismas muy poco restrictivas. La asignación de memoria para variables y constantes las hace directamente el traductor.

c) Dispone de instrucciones potentes, conteniendo *operadores y funciones* de una gran diversidad: aritméticas (seno, coseno, módulo, etc.), especiales (cambiar un dato de tipo real a entero, por ejemplo), lógicas (comparar, la función lógica Y, etc.), de tratamiento de caracteres (buscar una subcadena en una cadena de caracteres, por ejemplo), etc.

d) Pueden incluirse *comentarios* en las líneas de instrucciones, o en líneas específicas de comentarios. Esto facilita la *legibilidad* de los programas, tanto para el propio programador, como para otras personas.

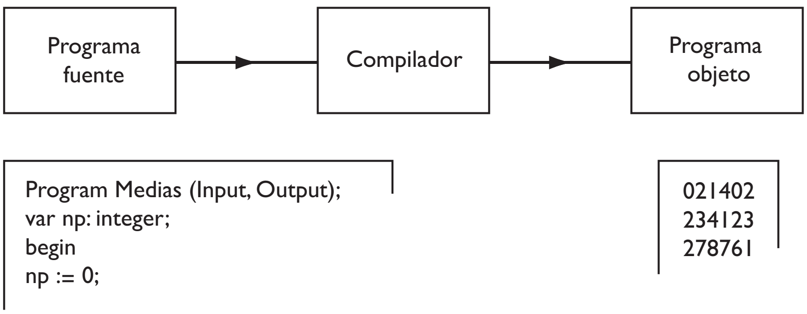


Figura 14.2. *Traducción de un programa escrito en un lenguaje **A** a otro lenguaje **B**.*

Como consecuencia de este alejamiento de la máquina y acercamiento a las personas, los programas escritos en lenguajes de programación no puede ser directamente interpretados por el computador, siendo necesario realizar previamente su *traducción* a lenguaje máquina. Hay dos tipos de traductores de lenguajes de programación, que vamos a considerar en las siguientes secciones: los compiladores y los intérpretes.

14.2 El proceso de traducción

Para facilitar el uso de los computadores se han desarrollado lenguajes de programación que permiten utilizar una simbología y una terminología próximas a las utilizadas tradicionalmente en la descripción de problemas. Como el computador puede interpretar y ejecutar únicamente código máquina, existen programas traductores, que traducen programas escritos en lenguajes de programación a lenguaje máquina. Un traductor es un programa que recibe como entrada un texto en un lenguaje de programación concreto, y produce, como salida, un texto en lenguaje máquina equivalente. El programa inicial se denomina **programa fuente**, y el programa obtenido, **programa objeto**. En la Figura 14.2 se representa un programa para obtener el valor medio de una serie de datos, redactado por el usuario en Pascal. Este programa fuente lo utiliza el traductor como conjunto de datos que debe procesar, y a partir de él genera como resultado el programa objeto.

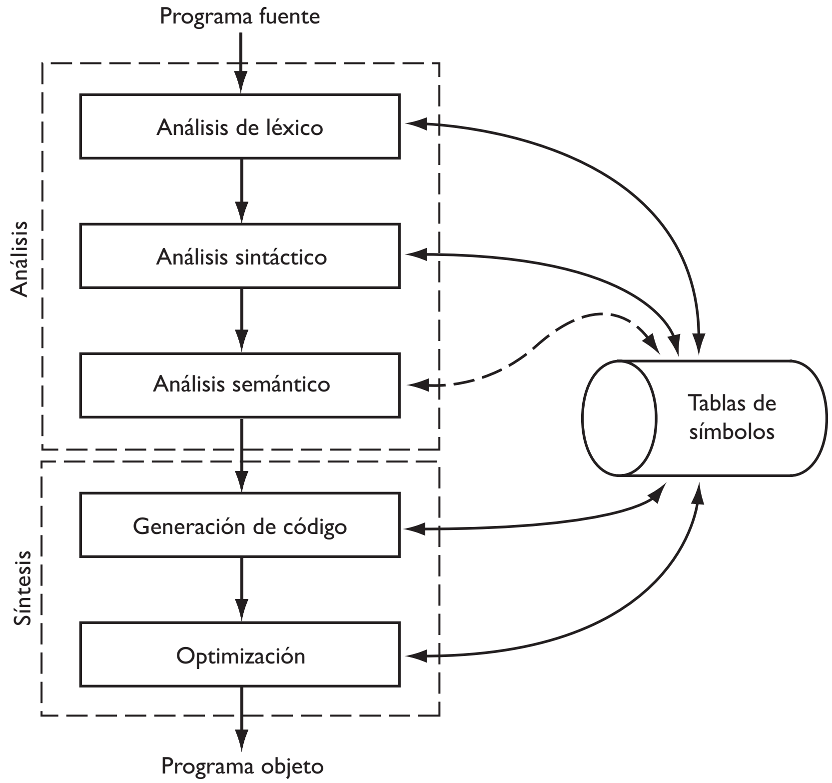


Figura 14.3. *Fases en la traducción de un programa fuente a un programa objeto, por un compilador.*

La traducción por un compilador (la **compilación**) consta de dos etapas fundamentales, que a veces no están claramente diferenciadas a lo largo del proceso: la etapa de análisis del programa fuente y la etapa de síntesis del programa objeto. Cada una de estas etapas conlleva la realización de varias fases. El análisis del texto fuente conlleva la realización de un análisis del léxico, de la sintaxis y de la semántica. La síntesis del programa objeto conlleva la generación de código y su optimización. Los siguientes epígrafes exponen cada una de estas fases.

La compilación es un proceso complejo y que consume a veces un tiempo muy superior a la propia ejecución del programa. En cualquiera de las fases de análisis el compilador puede dar mensajes sobre los errores que detecta en el programa fuente, cancelando en ocasiones la compilación para que el usuario realice las correcciones oportunas en el archivo fuente.

14.2.1 Análisis lexicográfico

Consiste en descomponer el programa fuente en sus elementos constituyentes o **símbolos de léxico** (*tokens*). Los símbolos de un lenguaje son caracteres o secuencias de caracteres que tienen un significado concreto en el lenguaje: cada una de las palabras reservadas, los símbolos de operadores, identificadores de variables, números, etc.

El **analizador lexicográfico** (o **escáner de léxico**) aísla los símbolos, identifica su tipo, y almacena en las tablas de símbolos la información del símbolo que pueda ser necesaria durante el proceso de traducción. Así, por ejemplo, convierte los números o constantes, que en el programa figuran en código de E/S (ASCII, por ejemplo), a su representación interna (entero, coma flotante, simple precisión, etc.), ya que esta información será necesaria a la hora de generar código (no bastará con saber que se multiplica por un número, sino que hay que saber su valor).

EJEMPLO 14.1

Al realizar el análisis léxico de la sentencia Pascal

Intensidad := (Magnitud + D) * 17

se generará la siguiente secuencia de símbolos:

Identificador [1]
Asignación
(
Identificador [2]
Operador +
Identificador [3]
)
*Operador **
Número [1]

Los números entre corchetes hacen referencia a la dirección del elemento en la tabla de símbolos. El contenido de la tabla de símbolos será el que se muestra en la Tabla 14.1.

Podemos considerar que, como resultado del análisis del léxico, se obtiene una representación del programa formada por la descripción de símbolos en las tablas, y una secuencia de símbolos junto con la referencia a la ubicación del símbolo en la tabla. Esta representación

contiene la misma información que el programa fuente, pero en una forma más compacta, no estando el código ya como una secuencia de caracteres, sino de símbolos. La información almacenada en las tablas de símbolos se completa, y utiliza, en las fases posteriores de la traducción.

El analizador de léxico realiza además otras funciones secundarias:

- Identifica, y salta, comentarios.
- Identifica, y pasa, espacios en blanco y tabulaciones.
- Informa de posibles errores de léxico. Dependiendo de la estructura de los analizadores de léxico y sintáctico, un error puede aparecer como error de léxico o como error sintáctico. Así, por ejemplo, si se da a una variable un nombre incorrecto (A%B en Pascal), el analizador de léxico puede, una vez leída la A, detectar que el siguiente carácter es incorrecto, generando un mensaje de error. No obstante es más habitual que el analizador de léxico interprete que A es un identificador (ya que éste no puede continuar con %) y, por tanto, aparezca un error sintáctico (al ser la construcción "Identificador % Identificador" incorrecta).

Tabla 14.1. *Contenido de la tabla de símbolos del ejemplo 14.1*

	Referencia	Nombre
Identificadores	1	Intensidad
	2	Magnitud
	3	D
	Referencia	**Valor**
Constantes	1	17

Aunque la programación de un analizador de léxico no es compleja, actualmente existen herramientas para generar analizadores de léxico, a partir de una descripción formal del léxico a reconocer.

14.2.2 Análisis sintáctico

La sintaxis de un lenguaje de programación especifica cómo deben escribirse los programas, mediante un conjunto de reglas de sintaxis o gramática del lenguaje. Un programa es sintácticamente correcto cuando sus estructuras (expresiones, sentencias declarativas, asignaciones, etc.) aparecen en un orden correcto. Así, en Pascal, por ejemplo, es sintácticamente incorrecta la sentencia:

$$A + B := (C - D) * 17$$

ya que antes de una asignación debe aparecer solamente un identificador. Una construcción puede ser sintácticamente correcta, y carecer de sentido (por ejemplo asignar un valor entero a una variable tipo cadena de caracteres).

Se han definido varios sistemas para definir la sintaxis de los lenguajes de programación (*metalenguajes*). Entre ellos cabe destacar la notación BNF (*Backus-Naur Form*) y los diagramas sintácticos. En ambos casos se definen las construcciones del lenguaje (*símbolos* no terminales), a partir de los símbolos terminales, reconocidos por el analizador de léxico. Una

construcción es una secuencia de símbolos con una estructura definida que tiene asignada un significado específico en el lenguaje (por ejemplo, la construcción formada por un identificador seguido del símbolo de asignación y de una expresión es una sentencia de asignación). Un diagrama sintáctico es un grafo dirigido, que describe, al recorrerlo, las distintas posibilidades de creación de una construcción del lenguaje. En BNF, la estructura de una construcción se representa por una expresión del tipo:

<Ciclo> ::= while <condición> do <bloque>

Las construcciones se pueden definir como secuencia de símbolos, repetición de símbolos o elección entre determinados símbolos. La Figura 14.4 muestra la representación usada, para cada una de estas estructuras, en diagramas sintácticos y en BNF. La Figura 14.5 muestra un ejemplo de la utilización de diagramas sintácticos para describir la sintaxis de algunas construcciones.

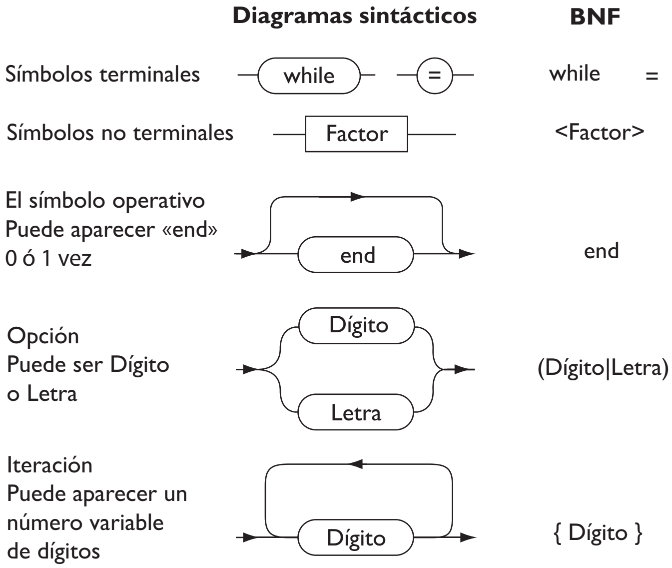


Figura 14.4. *Símbolos usados en Diagramas sintácticos (centro) y en notación BNF (derecha).*

Existe una gran diversidad de métodos para realizar el análisis sintáctico. A un nivel formal, la construcción de un analizador equivale a la construcción de un sistema software que admita como entrada tan solo las construcciones correctas en el lenguaje. Un método sencillo para construir un analizador es realizar un procedimiento para reconocer cada uno de los símbolos terminales del lenguaje. Cada procedimiento será una transcripción de un diagrama sintáctico, y comprobará si cada nuevo símbolo reconocido por el escáner puede ser correcto. Esto supone comprobar si el nuevo símbolo está dentro del conjunto de símbolos permitidos en cada momento, y decidir, en función del símbolo aceptado, cuál es el camino a seguir, en el diagrama sintáctico. Este tipo de análisis se conoce como descenso recursivo, dado que el análisis de construcciones recurrentes se realiza con procedimientos recursivos.

Con este esquema de análisis, el analizador puede generar mensajes de error cuando se encuentre un símbolo no esperado.

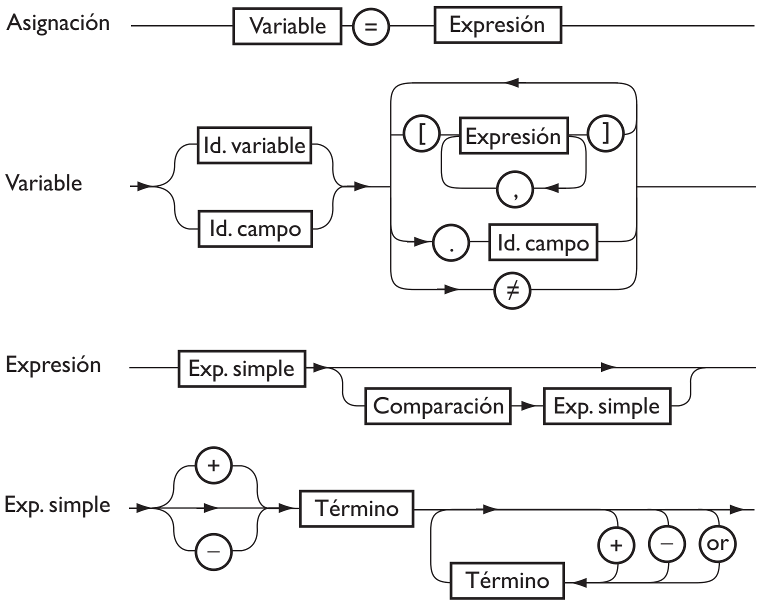


Figura 14.5. *Ejemplos de diagramas sintácticos.*

EJEMPLO 14.2

Para reconocer la construcción Expresión_simple de la Figura 14.5 se puede usar el siguiente procedimiento:

```
procedure Parse_Exp_Simple(char: siguienteSimbolo);
begin
if siguienteSimbolo = '+' or siguienteSimbolo = '−' then
        scaner(siguienteSimbolo);
parse_Termino(siguienteSimbolo);
while siguienteSimbolo in ['+','−','o'] do begin
        scaner(siguienteSimbolo);
        parse_Termino(siguienteSimbolo);
end;
```

donde se asume que el procedimiento scaner es el analizador de léxico, que devuelve el siguiente símbolo codificado en un carácter.

14.2.3 Análisis semántico

La semántica de un lenguaje de programación es el significado dado a las distintas construcciones sintácticas. El proceso de traducción es, en esencia, la generación de un código en lenguaje máquina con el mismo significado que el código fuente. En los lenguajes de programación, el significado está ligado a la estructura sintáctica de las sentencias. Así, una sentencia de asignación significa transferir el valor de la expresión de la derecha al identificador de la izquierda.

En el proceso de traducción, el significado de las sentencias se obtiene de la identificación sintáctica de las construcciones sintácticas y de la información almacenada en las tablas de símbolos.

Durante la fase de análisis semántico se pueden producir errores, cuando se detectan construcciones "sin un significado correcto". Por ejemplo, asignar a una variable definida como dato numérico en simple precisión el valor de una variable cadena de caracteres es semánticamente incorrecto para algunos compiladores.

14.2.4 Generación y optimización de código

En esta fase se crea un archivo con un código en lenguaje objeto (normalmente lenguaje máquina) con el mismo significado que el texto fuente. El archivo-objeto generado puede ser (dependiendo del compilador) directamente ejecutable, o necesitar otros pasos previos a la ejecución, tales como ensamblado, encadenado y carga (véase Sección 13.1). En algunas ocasiones se utiliza un lenguaje intermedio (distinto del código objeto final), con el propósito de facilitar la optimización del código.

En la generación de código intermedio se completan y consultan las tablas generadas en fases anteriores (tablas de símbolos, de constantes, etc.). También se realiza la asignación de memoria a los datos definidos en el programa.

La generación de código puede realizarse añadiendo procedimientos en determinados puntos del proceso de análisis, que generen las instrucciones en lenguaje objeto equivalentes a cada construcción en lenguaje fuente reconocida.

En la fase de optimización se mejora el código intermedio analizándose el programa objeto globalmente. Un programa puede incluir dentro de un ciclo, que debe ejecutarse diez mil veces, una sentencia que asigna a una variable un valor constante (**B=7.5**), no alterándose dicho valor (**B**) en el lazo (Figura 14.6a). Con ello, innecesariamente se asignaría el valor 7.5 a la variable B diez mil veces. El optimizador sacaría la sentencia B=7.5 fuera (antes) del bucle, ejecutándose así dicha instrucción una sola vez (Figura 14.6b). Hay que hacer notar que el programa inicial es correcto (con **B=7.5** dentro del lazo los resultados del programa son los mismos), pero la optimización realizada por el compilador reduce el tiempo de ejecución. Usualmente las optimizaciones se realizan en varias fases y sobre el código intermedio.

```
      DO 7 I = 1,10000
            R=37.–I*35.
            B = 7.5
7           Z = B–SIN(–R/35000)
```

(a)

```
      B = 7.5
            DO 7 I = 1,10000
                  R=37.–I*35.
7                 Z=B–SIN(–R/35000)
```

(b)

Figura 14.6. *El compilador optimizaría (reduciría) el tiempo de ejecución del bucle (a), sacando la sentencia B=7.5 fuera del lazo (b).*

Existen compiladores que permiten al usuario omitir o reducir, a su conveniencia, las fases de optimización, disminuyéndose así el tiempo global de la compilación.

14.3 Compiladores e intérpretes

Un **compilador** traduce un programa fuente, escrito en un lenguaje de alto nivel, a un programa objeto, escrito en lenguaje ensamblador o máquina. El programa fuente suele estar

contenido en un archivo, y el programa objeto puede almacenarse como archivo en memoria masiva para ser procesado posteriormente, sin necesidad de volver a realizar la traducción. Una vez traducido el programa, su ejecución es independiente del compilador, así por ejemplo, cualquier interacción con el usuario sólo estará controlada por el sistema operativo. (Figura 14.7)

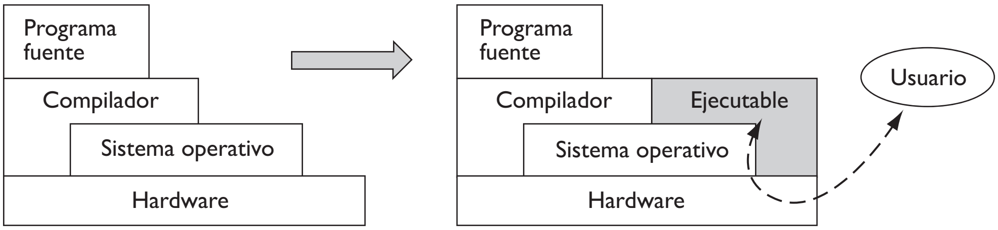


Figura 14.7. *Al compilar un texto fuente se crea un programa, cuya ejecución es independiente del compilador.*

Un **intérprete** hace que un programa fuente escrito en un lenguaje vaya, sentencia a sentencia, traduciéndose y ejecutándose directamente por el computador. El intérprete capta una sentencia fuente, la analiza e interpreta dando lugar a su ejecución inmediata, no creándose, por tanto, un archivo o programa objeto almacenable en memoria masiva para ulteriores ejecuciones. Por tanto, la ejecución del programa está supervisada por el intérprete (Figura 14.8).

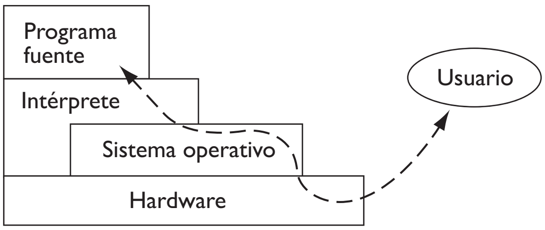


Figura 14.8. *Esquema de la ejecución de un programa con un intérprete.*

En la práctica el usuario crea un archivo con el programa fuente. Esto suele realizarse con un editor específico del propio intérprete del lenguaje. Según se van almacenando las instrucciones simbólicas, se analizan y se producen los mensajes de error correspondientes; así el usuario puede proceder inmediatamente a su corrección. Una vez creado el archivo fuente el usuario puede dar la orden de ejecución y el intérprete lo ejecuta línea a línea. Siempre el análisis antecede inmediatamente a la ejecución, de forma que:

a) Si una sentencia forma parte de un bucle, se analiza tantas veces como tenga que ejecutarse el bucle. Si el programa de la Figura 14.6a fuese traducido por un intérprete de FORTRAN, la sentencia **R=37.-I*35.** se analizaría 10.000 veces, y no una sola vez, como ocurriría con un compilador de FORTRAN.
b) Las optimizaciones sólo se realizan dentro del contexto de cada sentencia, y no contemplándose el programa o sus estructuras en conjunto. La optimización de la Figura 10.6b no podría ser realizada por un intérprete, entre otros motivos porque cuando llegue a **B=7.5** ya se han ejecutado las sentencias anteriores, no pudiendo, por tanto, ubicar la sentencia **B=7.5** antes del lazo.

c) Cada vez que utilicemos un programa tenemos que volver a analizarlo, ya que en la traducción no se genera un archivo objeto que poder guardar en memoria masiva (y utilizarlo en cada ejecución). Con un compilador, aunque la traducción sea más lenta, ésta sólo debe realizarse una vez (ya depurado el programa) y cuando deseamos ejecutar un programa ejecutamos el archivo objeto, que se tradujo previamente.

Los intérpretes, a pesar de los inconvenientes anteriores, son preferibles a los compiladores cuando el número de veces que se va a ejecutar el programa es muy bajo y no hay problemas de velocidad; además, con ellos puede ser más fácil desarrollar programas. Esto es así porque normalmente la ejecución de un programa bajo un intérprete puede interrumpirse en cualquier momento para conocer los valores de las distintas variables y la instrucción fuente que acaba de ejecutarse. Cuando esto se hace se tiene una gran ayuda para la localización de errores en el programa. Con un programa compilado esto no se puede realizar, salvo que el programa se ejecute bajo el control de un programa especial de ayuda denominado **depurador** (*debugger*). Además, con un interprete, cuando se localiza un error sólo debe modificarse este error y volver a ejecutar a partir de la instrucción en cuestión. Con un compilador, si se localiza un error durante la ejecución, el programador debe corregir las instrucciones erróneas sobre el archivo fuente (no sobre el objeto), y volver a compilarlo, comenzando la ejecución de nuevo.

Obviamente los intérpretes resultan más pedagógicos para aprender a programar, ya que se pueden detectar y corregir más fácilmente sus errores.

Otra ventaja adicional de los traductores-intérpretes es que ocupan, por lo general, menos memoria que los compiladores, pudiendo en ciertos casos estar grabados en memoria ROM (caso, por ejemplo, del BASIC en los microcomputadores).

Existen en la actualidad traductores (tal como el Turbo Pascal) que en cierta medida disponen de las ventajas tanto de los intérpretes como de los compiladores. Estos traductores se denominan **compiladores interactivos o incrementales**. Un compilador interactivo es un sistema que permite la edición, compilación, ejecución y depuración de programas escritos en un determinado lenguaje de alto nivel. Pueden presentar esencialmente las siguientes ventajas:

- Es muy cómoda la creación y depuración del programa, puesto que la iteración edición-compilación-ejecución no implica ejecutar distintos programas desde el sistema operativo.
- Permite que el compilador durante la edición realice parte de las comprobaciones, y en algunos casos la traducción a código intermedio, lo que permite detectar con antelación parte de los errores.
- Cuando se realiza una modificación puede no ser necesario retraducir todo el programa, si no tan solo la parte implicada en el cambio (cuando esto ocurre el traductor se dice que es un **compilador incremental**).

Existen lenguajes cuyos traductores se idearon como intérpretes (BASIC, APL, LISP, y PROLOG), y otros como compiladores (FORTRAN, COBOL, ALGOL, SNOBOL, PL/I, C, MODULA y ADA). No obstante, para un lenguaje dado, pueden existir tanto compiladores como intérpretes (este es el caso de los lenguajes BASIC, FORTRAN, LISP y SNOBOL, entre otros). Un caso notable es el de Java, que consigue una gran portabilidad gracias a la descomposición del proceso de traducción en dos fases. En la primera el programa se traduce a un código intermedio normalizado y, por tanto, portable. En la segunda fase este código es interpretado.

14.3.1 Traductores cruzados. Emuladores

Se denominan **traductores cruzados** a aquellos traductores que efectúan la traducción de programas fuente a programas objeto en un computador distinto (computador B) a aquel en el que se ejecutará el programa objeto (computador A). El computador B se denomina computador anfitrión (*host*) y el computador A, computador huésped (*guest*). Por lo general el computador anfitrión es más potente que el huésped.

También en un computador puede simularse el comportamiento de otro. Estos programas de simulación se suelen denominar **emuladores**. Existe, por ejemplo, un emulador para el microcomputador Apple Macintosh que utiliza como anfitrión el IBM-PC. Así un programa realizado para el Apple, sin modificación alguna, puede ejecutarse en el IBM PC. También hay emuladores del IBM-PC que actúan en el Apple.

14.4 Clasificación de los lenguajes

Los lenguajes de programación se pueden clasificar atendiendo a varios criterios. Un criterio simple es el nivel. El nivel hace referencia a lo próxima que la forma de expresar las sentencias esté al hombre (al lenguaje natural), o a la máquina (al lenguaje de ceros y unos de los circuitos electrónicos). Ya hemos hecho referencia a las diferencias entre los lenguajes de programación y los lenguajes de bajo nivel.

A un nivel más alto que el de los lenguajes de programación se encuentran los *lenguajes declarativos.* En los lenguajes declarativos, o de órdenes, los programas están formados por sentencias que ordenan “qué es lo que se quiere hacer”, no teniendo el programador que indicar al computador el proceso detallado (el algoritmo) de “cómo hacerlo”. En este grupo se incluyen ciertos lenguajes especializados, para propósitos muy específicos, tales como recuperación de la información en bases de datos (NATURAL e IMS), análisis de circuitos electrónicos, y realización de cálculos estadísticos (SAS, etc). La Figura 14.9 recoge algunos de los lenguajes de programación más relevantes, bien por su implantación o por su aportación conceptual.

Los lenguajes de alto nivel se pueden clasificar, atendiendo al estilo de programación, en dos grandes grupos:

a) Lenguajes basados en la asignación de valores (**Lenguajes Imperativos**, o **procedurales**). Se fundamentan en la utilización de variables para almacenar valores, y en la realización de operaciones con los datos almacenados. La mayoría de los lenguajes son de este tipo: FORTRAN, BASIC, COBOL, Pascal, Modula, ADA, C, etc.
b) Lenguajes basados en la definición de funciones o relaciones. No utilizan instrucciones de asignación (sus variables no almacenan valores). Los programas están formados por una serie de definiciones de funciones (**Lenguajes Funcionales**, como LISP) o de predicados (**Lenguajes de Programación Lógica**, como PROLOG).

Dentro de los lenguajes imperativos hay que destacar a los **lenguajes orientados a objetos**, implantados a lo largo de la década de los 80, y que introducen una nueva filosofía de programación. En ellos, los programas se descomponen en **objetos** que implementan partes del sistema encapsulando los datos que almacenan su estado junto con los métodos que se utilizan para acceder a ellos. De este modo, las distintas partes del programa son menos dependientes entre sí, lo que facilita el mantenimiento de las aplicaciones, al reducir la probabilidad de que una modificación local se propague por toda la aplicación. Actualmente existen versiones orientadas a objetos de muchos lenguajes de alto nivel (Objective C, C++, Object Pascal, PHP 5).

Fecha	Lenguaje	Tipo	Características
1956	FORTRAN	Procedural	Desarrollado para aplicaciones científicas.
1958	Algol	Procedural	Introduce el concepto de ámbito y la notación BNF.
1960	Cobol	Procedural	Orientado a gestión. Introduce los conceptos de registro.
1960	LISP	Funcional	Primer lenguaje funcional.
1964	Basic	Procedural	Lenguaje pequeño no estructurado, que habitualmente es interpretado.
1971	Prolog	Lógico	Primer lenguaje lógico.
1971	Pascal	Procedural	Introduce la comprobación de tipo estricta.
1971	Smalltalk	Orientado a objetos	Primer lenguaje orientado a objetos.
1973	C	Procedural	Es el lenguaje básico de los sistemas Unix. Se caracteriza por su economía de expresión Diseñado para programación de sistemas.
1979	Ada	Procedural	Desarrollado para reducir el coste de desarrollo y mantenimiento. Incorpora algunos mecanismos propios de lenguajes orientados a objetos.
1980	Modula-2	Procedural	Descendiente de Pascal, diseñado para permitir reusabilidad. Descompone los programas en módulos.
1982	C++	Orientado a objetos	Dialecto orientado a objetos de C. Mantiene la sintaxis y el estilo de C.
1987	Perl	Procedural	Lenguaje interpretado, orientado al procesamiento de texto. Permite el procesamiento de expresiones regulares.
1990	Python	Procedural	Lenguaje interpretado multi-paradigma.
1991	Java	Orientado a objetos	Lenguaje portable, fuertemente inspirado en C, desarrollado para programación en internet.
1995	PHP	Procedural	Lenguaje interepretado, derivado de Perl y orientado a la programación de aplicaciones web.

Figura 14.9. *Lenguajes de programación más relevantes.*

El aumento de potencia de los computadores ha hecho que la eficiencia deje de ser el factor predominante en la elección del lenguaje de programación para muchas aplicaciones. En aplicaciones en las que el tiempo de cálculo no es grande el tiempo de desarrollo y prueba se ha convertido en el factor determinado. Esto ha favorecido el desarrollo de lenguajes interpretados.

Otra tendencia reciente es la utilización de **herramientas visuales**, que tratan de sustituir parte del proceso de escritura del código, por realización de dibujos, o selección de iconos. Las herramientas visuales permiten realizar una descripción pictórica de la interfaz de usuario del programa a desarrollar, a partir de la cual generan automáticamente el código en el lenguaje de alto nivel. Actualmente estas herramientas permiten automatizar en gran medida la generación de la parte del programa que controla la interfaz con el usuario, que en la práctica supone una parte muy importante del código generado (con frecuencia es más de la mitad del tamaño del programa).

Existen lenguajes cuyo objetivo no es describir programas, si no describir documentos, imágenes, consultas a bases de datos, etc. En todos ellos se utilizan las mismas técnicas usadas para describir y traducir los lenguajes de programación.

14.5 Conclusión

En este capítulo se han presentado los conceptos básicos relacionados con los lenguajes de programación. Los lenguajes de programación facilitan la creación de programas, permitiendo su utilización en diferentes computadores. Para poder ejecutar un programa escrito en un lenguaje de alto nivel se debe de traducir, o interpretar, éste.

El traductor debe realizar un análisis de código fuente, durante el cual se pueden detectar errores en el código, y una síntesis del código objeto.

EJERCICIOS

14.1 Expresar el diagrama sintáctico de la Figura 14.5 en BNF.

14.2 Dibujar el diagrama sintáctico equivalente a la expresión BNF:

<Dirección> ::= ([<Nombre>] <Empresa> | <Nombre>) <Ubicación>

<Ubicación> ::= <Calle> <Número> [<Piso>] <Ciudad>

14.3 Describir, usando notación BNF, la construcción de una constante real, a partir de los dígitos y los signos de puntuación.

14.4 Desarrollar un algoritmo para calcular expresiones aritméticas que contengan números enteros, operaciones simples y paréntesis, leídas como cadena de caracteres que se interpretará de izquierda a derecha.
Realizar un escáner que identifique los operadores y constantes numéricas, un analizador sintáctico, que compruebe la corrección sintáctica de la expresión. Añadir acciones al analizador que efectúen las operaciones. ¿Pueden aparecer errores semánticos?

14.5 Detectar errores frecuentes en un lenguaje de programación concreto, y determinar el nivel en el que se producen.

14.6 Describir usando BNF el funcionamiento de un horno microondas.

14.7 Describir la sintaxis de las fórmulas de composición química. Redactar un analizador sintáctico para la descripción realizada.

14.8 Se ha diseñado un robot capaz de moverse dentro de un laberinto. El robot es capaz de girar a derecha e izquierda, avanzar, detectar si está frente a la salida y detectar si está frente a la pared. Diseña un lenguaje de programación para el robot que incluya las operaciones de movimiento, la función lógica "no" y la estructura de control mientras. Realiza un intérprete de este lenguaje que opere sobre una simulación del robot en un laberinto en pantalla.

14.9 Describir la sintaxis de expresiones aritméticas simples formadas por operadores aritméticos, paréntesis y números enteros. ¿Cuál es la semántica de una oración en este lenguaje?

14.10 Programar un intérprete para este lenguaje para el lenguaje del ejercicio anterior. ¿Cuál debe ser el resultado de la ejecución de una expresión?

BIBLIOGRAFÍA

[Aho90] Aho, A. V.; Sethi, R.; Ullman, J. D.: *Compiladores. Principios, técnicas y herramientas;* Addison Wesley, 1990.

[Bro00] Brookshear, J. R.: *Computer Science*; 6th edt. Addison Wesley, 2000.

[Nor00] Norton, P.: *Introducción a la computación,* 3ª ed. McGraw-Hill, 2000.

[Pra01] Prat, T. W.; Zelkowitz, M. V.: *Programming Languages. Desing and Implementation*. 4ª ed. Prentice Hall, 2001.

[San88] Sanchís Llorca, F. J.; Galán Pascual, G.: *Compiladores. Teoría y Construcción*. Paraninfo, 1988.

[Ral03] Ralston, A.; Reilly, E. D.; Hemmendinger, D.: *Encyclopedia of Computer Science*. 4ª ed. John Wiley & Sons, 2003.

[Tre85] Tremblay, J. P.; Sorenson, P. G.: *The theory and practice of compiler writing*. McGraw-Hill, 1985.

[Tuc01] Tucker, A. B.; Noonan, R.: *Programming Languages: Principles and Paradigms,* McGraw-Hill, 2001.

[Wat91] Watt, D. A.: *Programming languages. Concepts and paradigms*. Prentice Hall, 1991.

CAPÍTULO 15 Archivos

En este capítulo se aborda el estudio de archivos, que son estructuras de datos externas a los programas. Es decir, no están almacenados en memoria principal, sino en disco.

En este capítulo se define el concepto de archivo, y se describe como éstos se interrelacionan con los programas de los usuarios. Posteriormente se analizan los tipos fundamentales de organizaciones haciendo especial hincapié en la forma en cómo en cada una de ellas puede localizarse un registro individual. Seguidamente se indican los criterios que deben seguirse para optar por una organización determinada en el diseño de un archivo. Por último se muestran algunos de los métodos de ordenación y búsqueda de registros en archivos.

15.1 Concepto de archivo

Los datos que se encuentran en memoria masiva suelen organizarse en archivos. Un **archivo** o (fichero) *es un conjunto de información sobre un mismo tema, tratada como una unidad de almacenamiento y organizada de forma estructurada para la búsqueda de un dato individual* (véase Sección 13.2). Un **archivo** está compuesto de **registros** homogéneos que contienen información sobre el tema. Así por ejemplo, los datos de nómina del personal de una empresa (tema) se pueden almacenar en un archivo, en el que cada registro contendría los **campos** o datos de nómina de cada empleado.

La vida de todo archivo comienza cuando se crea, y acaba cuando se **borra**. Durante la vida del archivo se suelen realizar sobre él determinadas operaciones de **recuperación** o **consulta** y de **mantenimiento** o **actualización**. Estas operaciones las realizan programas específicos, que actúan al nivel de registro. Las operaciones de actualización que se pueden realizar sobre el archivo son:

- **Modificación** de un registro.
- **Eliminación o borrado** de un registro.
- **Inserción** de un registro.

La mayor parte de las operaciones de recuperación y actualización implican la realización de una **localización** o **búsqueda** de un registro concreto para luego actuar sobre él (leerlo, escribir o cambiar parte de él, borrarlo, etc.).

En la Tabla 15.1 se muestra un esquema de las operaciones básicas relacionadas con la gestión de archivos.

Los sistemas informáticos disponen en su software de programas de utilidad (véase Sección 13.1) para efectuar operaciones básicas con archivos (creación, borrado, copia o duplicado entre dispositivos de igual o distinta naturaleza, etc.). Estos programas, al igual que los módulos del sistema operativo y utilidades generales del software de un computador, se utilizan mediante un lenguaje de control. Existen también paquetes de programas específicos denominados genéricamente **sistemas de gestión de archivos** que permiten al usuario diseñar cómodamente archivos con determinadas estructuras y realizar recuperaciones y actualizaciones eficazmente.

Tabla 15.1. *Operaciones básicas con archivos.*

CREACIÓN	
RECUPERACIÓN o CONSULTA o LECTURA ("retrieval")	
ACTUALIZACIÓN	– modificación de registro – eliminación o borrado de registro – inserción de registro
BORRADO	
OTRAS	– duplicado o copia – clasificación, ordenación ("sort") – mezcla o intercalado o fusión ("merge")

Cuando un usuario utiliza archivos desde un programa escrito en un lenguaje de alto nivel, por lo general, el archivo no es manejado directamente por el propio programa, sino por el sistema operativo o por el software específico del computador para gestión de archivos (véase Sección 13.8). Este software se encarga de efectuar los accesos necesarios al dispositivo donde se encuentra el archivo y transfiere la información solicitada del archivo al programa, o a la inversa (véase Figura 15.1). Esto facilita que los programas sean **trasladables**, pues no tienen que hacer referencia a la forma específica de gestionar la información sobre el soporte, que puede ser diferente de un sistema a otro. Así, por ejemplo, un programa escrito en FORTRAN para un computador determinada y que utilice archivos, puede, con relativa facilidad, ser trasladado a otro computador aunque su memoria masiva esté estructurada físicamente de otra forma.

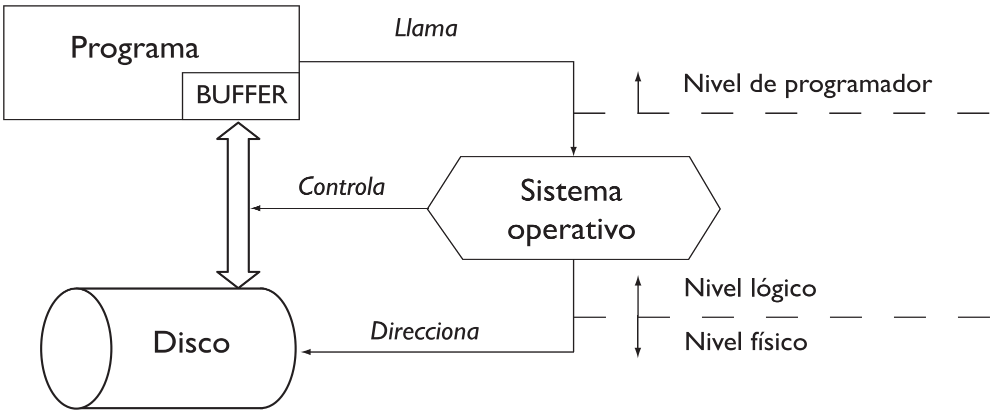


Figura 15.1 *Esquema de la gestión del acceso a archivos realizada por el sistema operativo.*

El sistema operativo transporta, cada vez que accede al dispositivo, una cantidad fija de información que depende de las características hardware o físicas de éste. Esta cantidad de información se denomina **bloque** o **registro físico**. En un bloque puede haber varios registros del archivo o puede que un registro ocupe varios bloques. Un factor importante en el diseño de archivos es la **longitud de bloque** o el **factor de blocaje** (o simplemente blocaje), que se define como el número de registros del archivo que entran en un bloque. Cuanto mayor sea éste, menor será el número de accesos al dispositivo necesarios para procesar el archivo.

El sistema operativo realiza además la transformación de la **dirección lógica** usada en los programas de usuario en la **dirección física** con la que se direcciona en el soporte. La dirección lógica (en adelante dirección) es la posición relativa que ocupa el registro en el archivo, tal y como se ve éste desde el programa del usuario. La dirección física es la posición real o efectiva donde se encuentra el registro en el soporte de información (dirección “hardware”). En el archivo los registros aparecen al usuario en *secuencia lógica*; es decir, ordenados linealmente. Normalmente este orden es relevante para la información que contiene. Por ejemplo, un archivo de nóminas puede estar ordenado alfabéticamente por apellidos y nombre de los empleados. No obstante en algunos casos el orden de los registros en el archivo puede no tener ninguna relación con la información que contiene.

El archivo es una estructura de datos externa al programa. El programa **accede** al archivo para leer, modificar o escribir un registro. Al leer se transfiere de bloque en bloque la información del archivo a un área o zona de la memoria principal asociada a las entradas/salidas del archivo (“buffer”), de forma que esta información sea procesable por el programa. El programa puede igualmente transferir información de esta área al archivo, modificando su contenido añadiéndole información.

Los archivos se guardan o almacenan en dispositivos de memoria masiva, estando limitados en tamaño tan solo por el de los dispositivos que los albergan. Los dispositivos o soportes de memoria auxiliar, (Capítulo 10) pueden ser de tipo secuencial o **no direccionables** o de acceso **directo** o **direccionables**. En los soportes no direccionables si el último acceso se ha efectuado al registro físico i, para acceder al registro j>i, hay que leer o pasar por los registros situados entre el i y el j. En los soportes direccionables, por el contrario, se puede leer o escribir directamente un registro físico sin más que dar su dirección física, sin necesidad de recorrer o leer otros registros.

Una **clave** o **identificativo** es un campo (o conjunto de campos) que identifica cada registro del archivo. En un archivo puede haber una, varias, o ninguna clave. Las claves se pueden utilizar para la localización rápida de los registros en archivos con determinadas organizaciones. Cuando una clave se utiliza como campo de localización en el archivo la denominaremos **llave**. En el caso de archivos de nóminas serían claves: el número de afiliación a la seguridad social, el nombre del empleado, su documento nacional de identidad y el número de registro de la empresa. Una clave se dice que es duplicada cuando puede tomar el mismo valor en dos o más registros distintos.

Los registros que componen el archivo pueden no tener la misma longitud. Esto puede suceder o bien por existir campos de longitud variable o bien por haber campos que se repiten un número variable de veces (o ambas cosas simultáneamente). Por lo que respecta a la longitud de sus registros, los archivos pueden estar formados por registros de uno de los siguientes tipos:

- **Longitud fija.** Todos los registros tienen la misma longitud.
- **Longitud variable.** El sistema reserva una palabra al comienzo de cada registro para anotar su longitud.
- **Delimitados.** El sistema incluye un carácter especial, normalmente el carácter de separación de líneas (“CR” o “NL”) al final de cada registro. En este caso se dice que el archivo es tipo **texto**.
- **Indefinido.** En este caso, el sistema operativo no realiza ninguna gestión sobre la longitud de los registros del archivo. Es el programa de usuario el que se encarga de localizar el principio y final de cada registro.

EJEMPLO 15.1

Supongamos un archivo que contiene información sobre los alumnos en una institución docente. Los registros, entre otros campos, contienen el nombre del alumno, y dirección y códigos de asignaturas en que está matriculado. Supongamos que el resto de los campos son de longitud fija. El nombre y dirección pueden definirse como campos de longitud fija; por ejemplo, de 32 y 54 caracteres, respectivamente. Como cada alumno puede estar matriculado en distinto número de asignaturas no todos los registros tendrían la misma longitud, a pesar de que todos los campos son de longitud fija.

El sistema operativo podría gestionar este archivo, o bien con registros de ***longitud variable*** *(en cuyo caso cada registro contendría un campo de control que indicaría el número de caracteres que contiene, o el número de asignaturas en que está matriculado el alumno correspondiente) o bien con registros* ***delimitados*** *(en cuyo caso el sistema operativo escribirá un carácter de control delimitador al final de cada registro).*

15.1.1 Tipos de archivos

En una aplicación informática se pueden utilizar archivos para realizar funciones diversas. Conocer la función que va a desempeñar un archivo concreto es fundamental a la hora de decidir como se debe organizar éste.

En una primera aproximación podemos clasificar los archivos en la forma que se indica en la Tabla 14.2.

Un **archivo permanente** contiene información relevante para una aplicación, es decir, los datos necesarios para el funcionamiento de la misma. Su vida es larga (al menos comparable a la de la aplicación para la que ha sido creado) y generalmente no puede generarse de una forma inmediata a partir de otros archivos.

Tabla 15.2. *Clasificación de los archivos según el uso que se hace de ellos.*

Archivos permanentes	– Archivos maestros – Archivos constantes – Archivos históricos
Archivos temporales	– Archivos intermedios – Archivos de maniobras – Archivos de resultados

Un archivo **temporal** contiene información relevante para un determinado proceso o programa, pero no para el conjunto de la aplicación. Se genera a partir de los datos de archivos permanentes o para actualizar éstos, y su vida es generalmente muy corta.

Los archivos permanentes se pueden clasificar en:

- **Archivos maestros**. Un archivo maestro contiene el estado actual de los datos susceptibles de ser modificados en la aplicación. Es el núcleo central de la aplicación. Todos los procesos están, en general, orientados a actualizar el archivo maestro o a obtener resultados de él. Un ejemplo de este tipo de archivo es el archivo de clientes de un banco, en el que los registros contienen información de identificación de clientes, su saldo en cuenta, etc.

- **Archivos constantes**. Un archivo constante es aquel que contiene datos fijos para la aplicación. En él las modificaciones son infrecuentes, normalmente se accede a él sólo para consultar. Serán archivos constantes los que contengan los intereses para distintos tipos de cuentas bancarias, la ubicación de estantes en una biblioteca, la capacidad de las aulas de un centro, una tabla de números primos, etc.
- **Archivos históricos**. Un archivo histórico es aquél que contiene datos que fueron actuales en tiempos anteriores. Se conservan para poder reconstruir la situación actual o situaciones anteriores. En algunos casos puede estar formado simplemente por los registros borrados del archivo maestro. Un archivo histórico puede contener, por ejemplo, los clientes que se han dado de baja en una entidad bancaria.

Los archivos temporales pueden ser:

- **Archivos intermedios**. Se utilizan para almacenar resultados de un programa que han de ser utilizados por otro, dentro de una misma tarea.
- **Archivos de maniobras**. Se utilizan para almacenar datos propios de un programa que éste no puede conservar en memoria principal, por falta de espacio en ésta. Se encuentran normalmente en programas de cálculo numérico, compiladores y editores. Su vida es siempre menor que el tiempo de ejecución del programa.
- **Archivos de resultados**. Se utilizan para almacenar datos elaborados que van a ser transferidos a un dispositivo de salida, como por ejemplo un **archivo de impresión**, que contiene datos que van a ser transferidos a una impresora.

EJEMPLO 15.2

*Supongamos que se desea almacenar y procesar los datos relativos a ocupación y cobros de un hotel. Será necesario en primer lugar almacenar una serie de datos fijos, como son las características de cada habitación, la tarifa de precios, etc. Esta información estará contenida en archivos, que por su naturaleza, serán **permanentes y constantes**. Los registros de estos archivos pueden tener la siguiente estructura:*

FDES: ***Archivo de descripción de habitaciones***

Campos: Nº de habitación, Nº de camas, Categoría, Orientación

FPRE: ***Archivo de precios***

Campos: Categoría de la habitación, Nº de camas, Precio en temporada alta, Precio en temporada baja

*Además, se deberán crear otros archivos que contendrán datos variables. Concretamente debe existir un archivo con información sobre el estado actual de ocupación del hotel, que será un archivo **permanente**, éste será el archivo **maestro** de la aplicación. Este archivo podrá tener la siguiente estructura:*

HOTEL: ***Archivo de ocupación del hotel***

Campos: Nº de habitación, Ocupada, Nº de personas, Apellidos y nombre, DNI, Fecha de llegada, Fecha de partida.

*Cada vez que un cliente abandona el hotel se creará un archivo de **resultados** conteniendo la factura que se debe cobrar a dicho cliente. Este archivo será transferido posteriormente a impresora, después de lo cual se borrará. Se trata por tanto de un archivo **temporal**. Los datos de ocupación de habitaciones del archivo Hotel pasarán, una vez dejada la habitación por el cliente, a un archivo **histórico**.*

15.2 Organización de archivos

Hay diferentes formas de estructurar u **organizar** los archivos sobre un soporte de información. Las características de utilización del archivo dependen de la organización que se adopte. Se debe pues optar por una u otra organización atendiendo a la forma en que se va a usar el archivo. Las principales organizaciones de archivos son:

- **Secuencial**. Los registros se encuentran en cierto orden yuxtapuestos consecutivamente. Los registros han de ser leídos necesariamente según este orden.
- **Indexada**. Se utiliza un índice para obtener la ubicación de la zona del archivo donde se encuentra del registro buscado. Esto permite localizar un registro sin leer previamente todos los que le preceden (sólo los de su zona).
- **Encadenada.** Cada registro contiene un puntero que permite localizar el siguiente registro. El archivo tiene la misma estructura que una lista lineal.
- **Directa**. La ubicación de registro en el soporte se obtiene a partir del valor de la llave (mediante un algoritmo de transformación de ésta).

Las siguientes secciones se dedican a analizar con más detalle estas organizaciones, y otras que surgen como combinación de ellas.

15.2.1 Organización secuencial

En esta organización los registros figuran almacenados contiguos (uno a continuación del anterior) siguiendo la secuencia lógica del archivo (Figura 15.2). Todas las operaciones que se realizan sobre el archivo se hacen según esta secuencia. Esta es la única organización de archivos susceptible de ser gestionada en un dispositivo no direccionable (soportes secuenciales tales como cinta magnética, cinta de papel o tarjeta perforada). La secuencia en que aparecen los registros en el archivo puede estar determinada por el valor de algún campo o ser simplemente temporal.

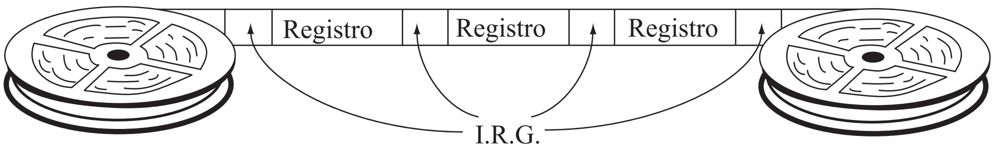


Figura 15.2. *Archivo secuencial en cinta magnética.*

Las distintas operaciones o acciones que se pueden realizar sobre archivos con esta organización son:

- **Añadir.** Sólo es posible escribir al final del archivo. La información se graba en el archivo escribiendo los registros uno a continuación de otro, en el orden en que se desea que estén en el archivo.
- **Consulta o recuperación.** La consulta se realiza en orden secuencial. Esto es, para leer el registro que ocupa la posición **n** en el archivo es necesario leer previamente los **n – 1** anteriores.
- **Inserción, modificación y eliminación.** No es posible realizar fácilmente ninguna de estas operaciones sobre un archivo secuencial. Si se necesita **actualizar** un archivo con organización secuencial se debe crear de nuevo el archivo. Esto se realiza por medio de un programa, escrito para tal fin, que utilizará la versión a modificar del archivo y un archivo

intermedio, también secuencial, del que leerá registros que escribirá en el nuevo archivo. Para esto es necesario que los registros aparezcan en el archivo ordenados según el valor de algún campo. Las modificaciones a realizar sobre el archivo se almacenarán previamente en el archivo intermedio antes mencionado (llamado usualmente archivo de **movimientos**) en la misma secuencia en que aparecen los registros en el archivo a modificar. Los registros del archivo de movimientos tienen la misma estructura que los del archivo a actualizar, más un campo, que normalmente se coloca al principio del registro, que contiene el código de la operación a realizar con el registro: modificar, eliminar o insertar.

EJEMPLO 15.3

Supóngase que el archivo maestro del Ejemplo 15.2 se ha construido con organización secuencial. El archivo debe ser actualizado incluyendo los nuevos clientes y retirando la información de los que ya han dejado el hotel. Para realizar estas modificaciones se procederá como sigue:

a) *Se crea un archivo de movimientos conteniendo las alteraciones a realizar. Sus registros pueden tener la misma estructura del archivo Hotel (Ejemplo 15.2), añadiendo el campo Tipo de alteración.*

El campo de tipo de alteración podrá tomar los siguientes valores:

Insertar: *La habitación no está en el archivo maestro y se pretende incluirla, se creará por tanto un nuevo registro en dicho archivo.*

Eliminar: *La habitación se borra del archivo maestro. El nuevo archivo maestro no contendrá el registro de esta habitación.*

Modificar: *Esta será la acción más frecuente. En ella o bien se altera un dato de la habitación o bien se da el alta o baja a un cliente. Si el programa de actualización realiza además alguna acción específica, como facturación, inclusión de información en un archivo histórico o confeccionar un parte de ingreso, será conveniente distinguir entre estos tres tipos de modificaciones.*

b) *En un momento concreto se ejecutará un programa de actualización que realizará las siguientes funciones:*

1. Ordena el archivo de movimientos según la misma secuencia que el archivo maestro. En este caso por orden de número de habitación.

2. Lee secuencialmente el archivo de movimientos. Para cada registro leído realiza las siguientes operaciones:

2.1. Busca secuencialmente en el archivo maestro la habitación leída del archivo de movimientos, hasta encontrar o bien ésta o bien una mayor. Cada registro leído en este proceso de búsqueda, salvo el último, será escrito en el archivo maestro nuevo.

2.2. Si el tipo de alteración es inserción y el registro ha sido localizado, dará un mensaje de error y escribirá el registro tal y como estaba. Si el registro no ha sido localizado escribirá en el archivo "maestro nuevo" el registro tal y como está en el archivo de movimientos.

2.3. Si el tipo de alteración es eliminación y el registro no ha sido localizado, da un mensaje de error. En caso contrario no escribe nada, ya que el registro debe ser borrado.

2.4. Si el tipo de alteración es modificación y el registro ha sido localizado escribe en el archivo de salida los valores dados en el archivo de movimientos ya que se desea modificar el valor del registro. Si en este caso no se localiza el registro se da un mensaje de error.

c) *Además el programa de actualización podrá realizar otras funciones con el registro, tal como se ha mencionado anteriormente.*

Después de esto habrá dos archivos maestros (véase Figura 15.3). El archivo maestro actualizado y el archivo maestro de partida. El archivo maestro no actualizado y el archivo de movimientos podrán borrarse o guardarse como información histórica. La organización secuencial no es la mejor organización para este archivo.

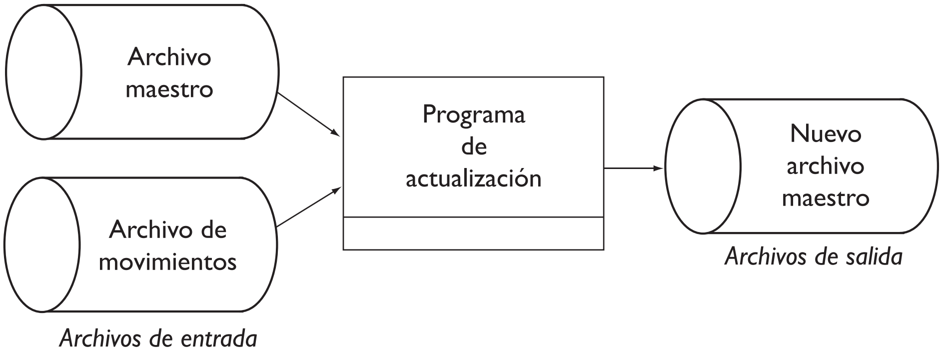


Figura 15.3. *Actualización de un archivo secuencial.*

Si el archivo se encuentra sobre un soporte direccionable (disco por ejemplo) es posible realizar algunas actualizaciones sobre el archivo secuencial, sin necesidad de crear otro archivo maestro como se hizo anteriormente (véase Figura 15.3). En este supuesto se podrán realizar las siguientes operaciones sobre el archivo, además de las ya descritas:

- **Modificación.** Una vez localizado un registro se puede reescribir éste en el propio archivo, siempre que al modificar el registro no aumente su longitud.
- **Borrado.** No es posible eliminar un registro del archivo. No obstante es posible marcarlo de tal forma que al leer se identifique como no válido, esto es lo que se llama **borrado lógico**. El registro se marca reescribiéndolo con un valor determinado en un campo especial que indica si el registro está borrado o bien con un valor no válido en un campo de información del registro. Así, por ejemplo, en el archivo HOTEL del Ejemplo 15.3 se puede utilizar como marca de borrado el valor –1 en el número de habitación. En cualquier caso el registro sigue estando en el archivo y por tanto ocupa espacio.
- **Consulta.** Si los registros son de longitud fija es posible determinar la dirección de comienzo de cada uno a partir de su posición relativa en el archivo. Se puede pues acceder a un registro conociendo su posición relativa en el archivo. En este caso, y si el archivo está ordenado según el valor de algún campo, es posible consultar en un registro conocido el valor del campo que se usa como llave, sin necesidad de leer el archivo secuencialmente. Para ello se deben usar algoritmos de búsqueda especiales (Sección 15.4.2).

La organización secuencial es adecuada para archivos en los que no se realizan procesos interactivos, y en los que en cada proceso se debe acceder a la mayor parte de los registros del archivo. Presentan la ventaja de aprovechar bien el espacio, ser sencillos de utilizar y de poderse utilizar con dispositivos secuenciales (cinta magnética) que son baratos.

15.2.2 Organización secuencial encadenada

En un archivo con organización **secuencial encadenada** junto a cada registro se almacena un puntero con la dirección del registro siguiente, según el orden lógico del archivo. Los archivos con esta organización sólo pueden ser gestionados, como es lógico, en soportes direccionables. Estructuralmente un archivo secuencial encadenado es equivalente a una lista lineal de registros. Las distintas operaciones se realizan sobre el archivo del siguiente modo:

- **Recuperación o consulta.** La consulta es secuencial, al igual que en un archivo con organización secuencial pura. Cada vez que se lee un registro se lee la posición del siguiente, lo que permite seguir la secuencia lógica del archivo.

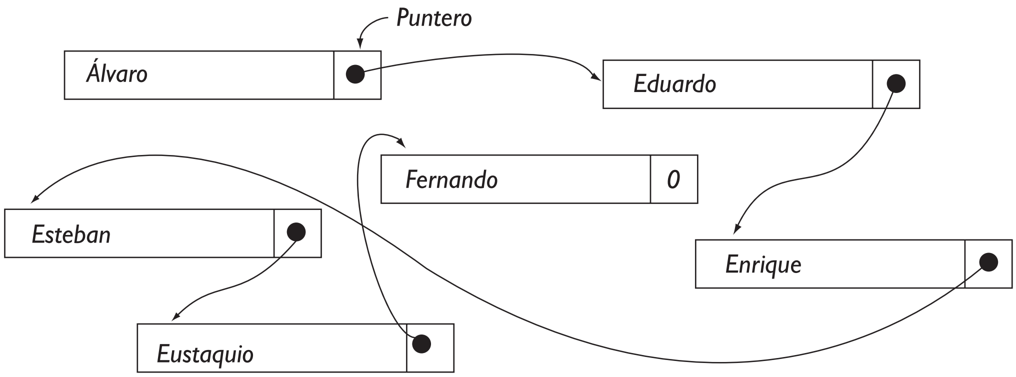


Figura 15.4. *Esquema de un archivo secuencial encadenado.*

- **Inserción**. Para insertar un registro es necesario en primer lugar localizar la posición en que se desea insertar; esto es, entre qué dos registros se quiere que aparezca al leer el archivo. Físicamente el registro se escribe en una zona libre, con el mismo valor de puntero que el registro que le va a preceder. Por último se modifica el registro anterior para actualizar el valor de su puntero, de forma que contenga la dirección del nuevo registro, véase Figura 15.5.
- **Borrado**. Un registro deja de estar en la secuencia de lectura del archivo cuando se elimina su dirección del puntero del registro anterior. Para borrar un registro se asigna al puntero del registro anterior la dirección del registro siguiente al que se desea borrar. El sistema operativo puede o no liberar el espacio ocupado por el registro. En el primer caso dicho espacio puede ser ocupado por información perteneciente a otro archivo. En ambos casos ese espacio podrá ser ocupado por otro registro del mismo archivo.
- **Modificación**. Si la modificación no implica un aumento de longitud del registro, éste puede reescribirse en el mismo espacio. En el caso de que el registro aumente de longitud se debe insertar el registro y posteriormente borrar la versión anterior a la modificación.

1	Alba	4
2	Elefante	3
3	León	5
4	Coral	2
5	Mosca	8
6	Ñu	0
7	Pantera	6
8	Morsa	7
9		
10		

(a) Localización del registro anterior

1	Alba	4
2	Elefante	3
3	León	5
4	Coral	2
5	Mosca	8
6	Ñu	0
7	Pantera	6
8	Morsa	7
9	**Gato**	3
10		

(b) Escritura del nuevo registro

1	Alba	4
2	**Elefante**	9
3	León	5
4	Coral	2
5	Mosca	8
6	Ñu	0
7	Pantera	6
8	Morsa	7
9	Gato	3
10		

(c) Modificación de punteros

Figura 15.5. *Inserción en un archivo secuencial encadenado. Se inserta un registro con llave Gato.*

Si la longitud de los registros es mucho menor que el tamaño del bloque del dispositivo (factor de blocaje mayor que uno), en cada acceso a disco se podrán leer o escribir varios registros. Si en cada nodo de la cadena hay un solo registro (tal como se ha expuesto antes) se ralentiza el acceso, ya que la lectura de cada registro implicará un acceso al dispositivo. Para reducir el tiempo de acceso se deben almacenar en cada bloque tantos registros como sea posible, encadenando los bloques, en lugar de encadenar los registros. Los registros contenidos en cada bloque serán consecutivos según el orden lógico del archivo. El precio pagado por esta mejora es tener que usar un algoritmo de inserción más complicado, tal como el que se muestra en la Figura 15.6.

1	Asno	Coral	5
2	Pantera		3
3	Ñu		2
4	Mosca	Morsa	0
5	Elefante	León	4
6			
7			

Localización

1	Asno	Coral	5
2	Pantera		3
3	Ñu		0
4	Mosca	Morsa	2
5	Elefante	León	4
6	**León**		**4**
7			

Creación de espacio en el registro

1	Asno	Coral	5
2	Pantera		3
3	Ñu		0
4	Mosca	Morsa	2
5	**Elefante**	**Gato**	6
6	León		4
7			

Escritura del registro

Figura 15.6. *Inserción en un archivo secuencial encadenado con varios registros por bloque. Se inserta un registro con llave Gato.*

Los archivos con organización encadenada son útiles en aquellos casos en que se deban realizar frecuentes inserciones de pocos registros. Si cada vez que se accede al archivo para insertar se deben insertar muchos registros, en comparación con el número de registros del archivo, puede ser preferible utilizar un archivo secuencial puro. La principal ventaja de la organización secuencial encadenada es su flexibilidad, su principal inconveniente es, al igual que en la organización secuencial pura, su limitación a consulta secuencial.

15.2.3 Organización secuencial indexada

Un archivo con organización secuencial indexada está formado por dos estructuras o zonas distintas: zona de registros, y zona de índices. En la **zona de registros** se encuentran los registros ordenados según el valor de una llave. En esta zona se pueden direccionar los registros. Formalmente su estructura es semejante a la de un archivo con organización aleatoria y direccionamiento directo o a un archivo secuencial puro en que se pueda direccionar al nivel de registro. La **zona de índices** se procesa sólo de forma secuencial. Su estructura es la de un archivo secuencial puro en que cada registro contiene dos campos: **llave y dirección** de un registro.

La zona de registros está dividida en tramos lógicos. Cada **tramo** está formado por una serie de registros consecutivos. Por cada tramo de la zona de registros hay un registro en la zona de índices. En este registro se encuentra el valor mayor de llave del tramo (valor de llave del último registro del tramo) y la dirección del primer registro del tramo (véase Figura 15.7). Ambas zonas pueden o no estar en un mismo archivo del sistema. En cualquier caso la gestión de la estructura la realiza el sistema operativo o un paquete software especial. Por tanto el usuario de esta estructura no necesita conocer la existencia de ambas zonas, pudiendo contemplar ambas como un todo.



Figura 15.7. *Esquema de un archivo secuencial indexado.*

Con archivos secuenciales indexados se pueden realizar las siguientes operaciones a nivel de registro:

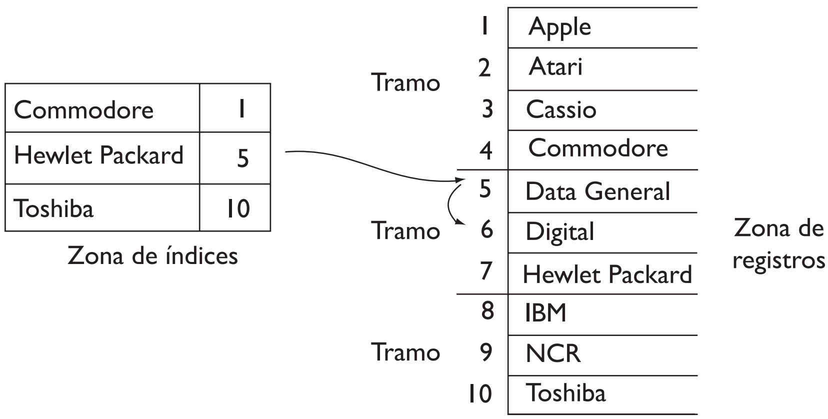


Figura 15.8. *Consulta de un registro por llave, se busca el registro con llave "Digital".*

- **Consulta.** Con este tipo de archivos se pueden realizar consultas secuenciales, tal como se explicó para archivos con organización secuencial. Además esta organización de archivo permite realizar consultas por llave (esto es, localizar un registro conocida su

llave) sin necesidad de leer los registros que le anteceden en al archivo (Figura 15.8). El procedimiento a seguir para realizar una consulta por llave es:

1. Leer secuencialmente las llaves en la zona de índices hasta encontrar una mayor o igual a la del registro buscado. Obtener la dirección de comienzo del tramo donde se encuentra el registro.
2. Leer secuencialmente en la zona de registros a partir de la dirección obtenida en la zona de índices hasta encontrar el registro buscado o uno con valor de llave mayor que el buscado. En este último caso el registro no se encuentra en el archivo.

- **Inserción.** Dado que ambas zonas son secuenciales no es posible insertar un registro en archivos con esta organización. En algunos casos se permite la escritura de nuevos registros al final de la zona de registros. Estos registros, como es lógico, no podrán ser consultados por llave con el procedimiento antes descrito.
- **Eliminación y modificación.** Al estar los registros escritos en secuencia no es posible borrar un registro. La única forma de eliminar la información contenida en un registro es marcándolo, lo que se conoce como *borrado lógico*. Las modificaciones son posibles tan sólo si el registro no aumenta de longitud al modificarlo y no se altera el valor de la llave de éste.

El tiempo de localización de un registro depende de la estructura de la zona de índices. Se puede demostrar que el mínimo número de accesos se obtiene cuando se verifica:

$$N_t = \sqrt{\frac{R \cdot B_r}{B_i}} \; ; \; I = \sqrt{\frac{R \cdot B_i}{B_r}}$$

siendo:

N_t el número de registros en cada tramo.

R el número total de registros.

I el número de registros en la zona de índices.

B_i el factor de blocaje de la zona de índices.

B_r el factor de blocaje de la zona de registros.

Si la tabla de índices no es demasiado grande y las consultas son frecuentes se puede cargar ésta en la memoria principal, con la consiguiente disminución del tiempo de búsqueda. Otra forma de disminuir el tiempo de búsqueda es utilizar varias tablas de índices en cascada. La Figura 15.9 muestra un archivo con dos niveles de índice. En cualquier caso la estructura no debe ser demasiado compleja, si las actualizaciones son frecuentes, pues habrá que recrear el archivo, siguiendo el proceso descrito para archivos secuenciales puros.

Esta organización resulta especialmente útil cuando se debe combinar consultas a registros concretos y el procesamiento secuencial de todo archivo. Su principal inconveniente es la imposibilidad de realizar actualizaciones.

Para permitir actualizaciones es necesario incluir en la estructura una zona de **desbordamientos**, tal como se vio anteriormente. En la zona de desbordamientos (o zona de *overflow*) los registros están desordenados, cada nuevo registro se escribe el final de ésta. Esto complica la búsqueda por llave, pues, si el registro no es encontrado en la zona de registros por el algoritmo antes descrito, es necesario buscarlo secuencialmente en la zona de desbordamientos. Además se imposibilita la consulta secuencial del archivo, ya que los registros no aparecen ordenados por llave.

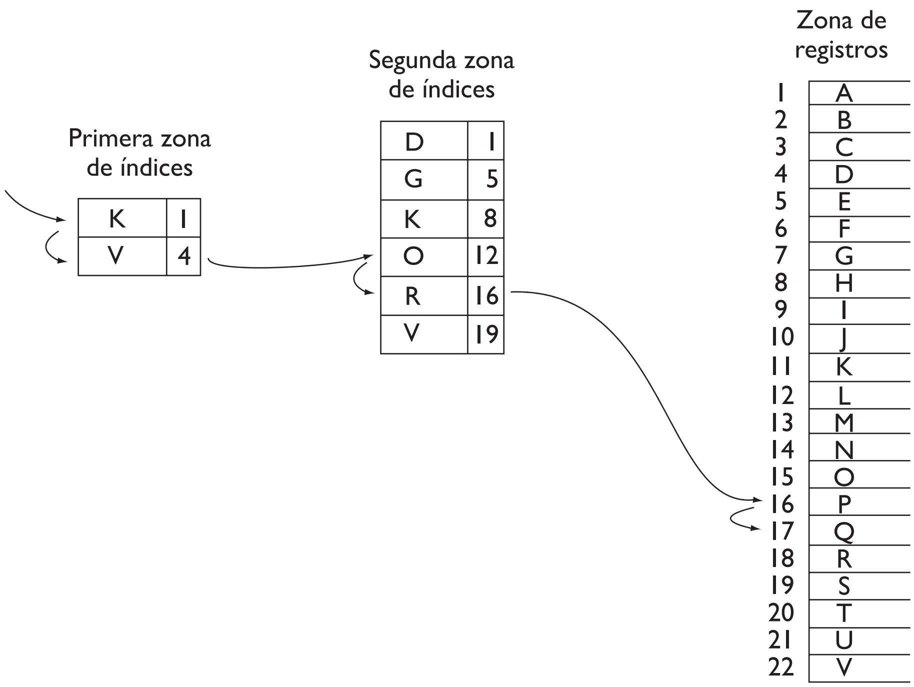


Figura 15.9. *Archivo secuencial indexado con dos niveles de índices.*

Otra mejora de la estructura anterior es incluir punteros entre los registros de forma que éstos mantengan el orden lógico de los registros. A esta organización se le llama **secuencial indexada encadenada**. Cuando se crea un archivo con esta organización su estructura es igual a la de un archivo secuencial indexado, salvo que se ha previsto un campo en cada registro para albergar un puntero.



Figura 15.10. *Archivo con organización secuencial indexada encadenada.*

Para insertar un nuevo registro es necesario encontrar el que le sigue en la zona de registros. Se escribe el nuevo registro en la zona de desbordamientos, y se reescribe el siguiente según el orden lógico, para incluir el puntero al registro recién grabado.

La eliminación de un registro debe realizarse por marcas. Esto no implica en ningún caso la realización de modificaciones en el índice, pero degrada al archivo. La consulta es semejante a la realizada para el caso no encadenado. Esta organización reúne las ventajas de la organización secuencial indexada y de la organización secuencial encadenada. Cuando el número de registros borrados es grande o las cadenas de desbordamiento son largas el archivo se degrada, esto es, su estructura deja de ser eficaz. Cuando se llega a este punto es necesario reorganizar el archivo; esto es, volver a crearlo introduciendo los registros secuencialmente según el valor de clave.

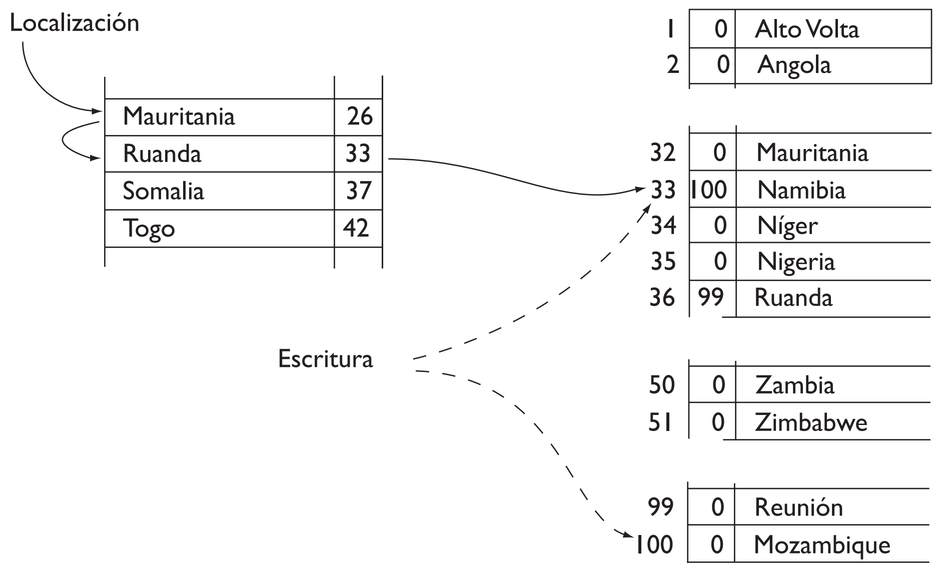


Figura 15.11. *Inserción en un archivo secuencial indexado encadenado. Se inserta "Mozambique".*

15.2.4 Organización directa o aleatoria

Un **archivo con organización directa o aleatoria (*random*)** *es un archivo escrito en un soporte direccionable para el cual existe una transformación conocida que genera la dirección de cada registro en el archivo a partir de una llave*. El nombre de "aleatorio" se debe a que normalmente no existe ninguna vinculación aparente entre el orden lógico de los registros y su orden físico. No resulta adecuado, en la mayoría de los casos, realizar una consulta secuencial en un archivo aleatorio, ya que los registros no serían leídos según el orden de la llave.

El problema fundamental de esta organización es la elección de la transformación o **método de direccionamiento** que se ha de usar. En algunos casos aparecen dos situaciones no deseadas:

- Hay direcciones que no corresponden a ninguna llave, y por tanto zonas de disco sin utilizar.
- Hay direcciones que corresponden a más de una llave. En este caso se dice que las llaves son **sinónimas** para esa transformación.

EJEMPLO 15.4

Una empresa de servicios desea controlar sus relaciones con sus clientes. Se decide para ello almacenar los datos de estos en un archivo con organización directa. El número de clientes es del orden de mil, utilizándose como llave el D.N.I. del cliente. Si se utiliza como transformación la extracción de los cuatro últimos dígitos del D.N.I. es posible generar la dirección de cualquier registro:

*Llave: 27 94**7 512*** *dirección: 7512*

En este caso es evidente que quedarán direcciones libres pues el archivo contendrá sólo mil registros de información y ocupará diez mil. Además es posible que existan sinónimos, tales como los siguientes: 25 917 512, 28 077 512 y 37 997 512, que dan lugar a la misma dirección.

Hay dos formas de resolver el problema de los sinónimos, siempre a costa de complicar la estructura del archivo:

1. Cuando se asocia a una llave una dirección ya ocupada por un registro distinto (esto, es por un sinónimo de esta llave), se busca en el archivo hasta encontrar una posición libre donde escribir el registro.
2. Se reserva una **zona de desbordamientos** donde se escribirán los registros que no se pueden escribir en la posición que les corresponde según la transformación, en la que denominaremos zona principal.

Si se utiliza el primer procedimiento, la búsqueda de una posición libre puede realizarse secuencialmente a partir de la posición asignada, o aplicando a la dirección obtenida un segundo método de direccionamiento. En cualquier caso ambos procedimientos son lentos y degradan considerablemente el archivo. Si en el Ejemplo 15.4 el registro con llave 25 917 512 se desborda a la dirección 7513, y después intentamos escribir un registro con llave 37 457 513, éste se deberá desplazar por estar la posición que le corresponde ocupada por un registro al que a su vez no le pertenece dicha posición.

Si se utiliza **zona de desbordamientos** se puede gestionar ésta bien secuencialmente o bien encadenada a la zona principal. Esto último presenta el inconveniente de tener que reservar un puntero en cada registro, pero permite un acceso más rápido a los sinónimos, lo que en general será más importante que el espacio.

Hay tres métodos básicos de direccionamiento para los archivos de organización directa:

- **Directo**. Se utiliza como dirección la propia llave.
- **Asociado**. A cada llave se le asocia una dirección mediante una tabla.
- **Calculado** o **por transformación de llave**. La dirección se obtiene realizando operaciones y transformaciones con la llave.

A continuación se describen brevemente estos tres tipos de direccionamientos para archivos de organización directa.

15.2.4.1 Direccionamiento directo

El direccionamiento directo sólo es factible cuando la llave es numérica y su rango de valores no es mayor que el rango de direcciones en el archivo. Se utiliza como dirección la propia llave.

En algunos casos pueden quedar lagunas de las direcciones sin utilizar, en lugares conocidos de antemano. En este caso se pueden ocupar dichas direcciones desplazando las direcciones superiores. Un archivo aleatorio con direccionamiento directo está siempre ordenado.

El archivo de habitaciones de un hotel (FDES en el Ejemplo 15.1) puede organizarse en forma aleatoria con direccionamiento directo, sin más que hacer la dirección igual al número de habitación.

15.2.4.2 Direccionamiento asociado

El direccionamiento **asociado** se puede utilizar para cualquier tipo de llave. Si se utiliza este método debe construirse una tabla en la que figurarán todas las llaves y la dirección donde se encuentra el registro correspondiente.

Al añadir nuevos registros las llaves se colocan al final de la tabla. Ésta se encontrará normalmente desordenada. Por tanto, habrá que localizar las llaves en ella por lectura secuencial, lo que ralentiza el acceso, al menos que la tabla esté residiendo en la memoria principal, se indexe, o se ordene y se busque en ella la llave con un procedimiento rápido, tal como el de "búsqueda dicotómica" (Sección 15.4.2).

Llave	Dirección
Argentina	1028
Venezuela	8798
Brasil	3451
Ecuador	
Panamá	64
Colombia	786
Chile	7

Figura 15.12. *Ejemplo de tabla para direccionamiento asociado.*

15.2.4.3 Direccionamiento calculado (hashing)

En el **direccionamiento calculado** (*hashing*), la dirección de cada registro se obtiene, o bien evaluando una expresión que utiliza como dato la llave, o bien realizando una transformación sobre dicha llave. Se utiliza cuando la llave no es numérica o, siendo numérica, cuando puede tomar valores dentro de un conjunto demasiado grande como para poder ser utilizada directamente como dirección. Cualquiera que sea la transformación usada es normal que aparezcan **sinónimos**, si bien la frecuencia con que estos aparecerán dependerá en cada caso concreto del algoritmo de transformación utilizado.

Si la llave no es numérica se utilizará una conversión previa para obtener un número a partir de ella. Usualmente se utiliza el equivalente decimal al propio código binario del carácter. Así, por ejemplo, en código ASCII al carácter **A** le correspondería el número 65 (véase Apéndice).

Existen gran cantidad de métodos de cálculo de dirección. Aquí sólo citaremos los más relevantes.

- **División**. La dirección es el resto de dividir la llave por una constante. Normalmente esta constante es el mayor de los números primos menores que el número de posiciones del archivo.
- **Extracción**. Consiste en utilizar como dirección un grupo de cifras de la llave, contiguas o no. En el Ejemplo 15.4 se muestra la utilización de este método.
- **Elevación al cuadrado**. Se utiliza cuando la representación numérica de la llave no es muy larga. Consiste en elevar ésta al cuadrado y tomar los dígitos centrales. Así, si hay 1.000 registros en el archivo y la llave es 3489:

  1º se calcula 3489^2, que es 12**3**121

  2º se toma como dirección las tres cifras centrales **3**
- **Plegamiento**. Se utiliza para llaves muy largas. Se descompone la llave en trozos de cifras contiguas del mismo tamaño y se suman éstos. Sobre el número generado se puede aplicar otro método si es necesario.

EJEMPLO 15.5

Si se utiliza un plegamiento en grupos de tres dígitos, se pueden direccionar con llaves de cualquier longitud. Así, por ejemplo para la llave 127448351 se obtendría la descomposición 12 744 831 751, que generará la siguiente dirección:

$$12 + 744 + 831 + 751 = 2338$$

Siempre que deba accederse a una posición, ya sea para introducir por primera vez un registro o para su consulta o modificación, debe aplicarse a la llave el algoritmo de transformación.

15.2.4.4 Operaciones básicas sobre un archivo con organización directa

Con un archivo con organización directa se pueden realizar todas las operaciones descritas. A diferencia de otras organizaciones es necesario realizar un proceso especial de creación del archivo. Ésta implica la elección de uno de los procedimientos de direccionamiento descritos en los epígrafes precedentes, y de un método de gestión de sinónimos, así como reservar espacio en disco para la zona principal.

- **Consulta**. La consulta se realiza por llave. Para leer un registro debe aplicarse a la llave el algoritmo de transformación y posteriormente leer el registro en la dirección resultante. Si el registro con la llave buscada no se encuentra allí se procederá, en función de cómo se haya resuelto gestionar los sinónimos, como se vio en epígrafes anteriores. La consulta secuencial no siempre es factible.
- **Borrado**. Siempre se realiza un borrado lógico, pudiéndose reutilizar el espacio del registro eliminado.
- **Modificación** e **inserción**. Siempre se puede modificar o insertar un nuevo registro, realizando la transformación de llave correspondiente.

La organización directa es útil para archivos donde los accesos deben realizarse por llave, accediéndose siempre a registros concretos. Si la información se va a procesar en conjunto con frecuencia puede ser más rentable una organización secuencial indexada.

15.3 Parámetros de utilización de un archivo

La organización de un archivo debe elegirse teniendo presente la utilización que va a hacerse del mismo. Ninguna organización es siempre mejor que otra, pero para cada caso concreto hay una ideal. Para caracterizar la utilización que se hace del archivo se utilizan los siguientes parámetros, que se describen a continuación: volumen, actividad, volatilidad y crecimiento.

- **Volumen**. Es el espacio, en caracteres, que ocupa el archivo en el soporte. Puede estimarse el volumen, antes de crear el archivo, multiplicando el número previsto de registros por su longitud media. Si el volumen es un problema crítico interesa usar organizaciones que aprovechen el espacio.
- **Actividad**. Caracteriza la importancia de los procesos de consulta y modificación del archivo. Se especifica con los siguientes parámetros:
  - **Tasa de consulta o modificación**. Es el porcentaje de registros consultados o modificados en cada tratamiento del archivo, respecto al total de registros en el archivo.
  - **Frecuencia de consulta o modificación**. Es el número de veces que se accede al archivo para consultar o modificar en un periodo de tiempo fijo (mes o año, por ejemplo). En algunos casos se determina la frecuencia de consulta de cada registro del archivo o de cada campo, para optimizar la distribución de los registros en el archivo o de los campos en el registro.
- **Volatilidad**. Es una indicación del peso de los procesos de inserción y borrado en el tratamiento del archivo. Se especifica con los parámetros **tasa de renovación** y **frecuencia de renovación**, con los mismos significados anteriores.
- **Crecimiento**. Caracteriza la variación de volumen del archivo. Se mide mediante la **tasa de crecimiento** que es el porcentaje de registros en que aumenta el archivo en cada tratamiento.

15.4 Métodos de ordenación y búsqueda

Son bastantes los procesos realizados sobre archivos, u otras estructuras de datos, que requieren que éstos se encuentren ordenados. Un archivo está **ordenado** (*sorted*) cuando *los registros aparecen en él de forma creciente o decreciente según el valor de uno o varios campos*.

Si el archivo está ordenado y se encuentra sobre un soporte direccionable, de tal modo que sea posible leer un registro sabiendo su número de orden en el archivo, es posible **localizar o buscar** con gran rapidez un registro que contenga un determinado valor en el campo por el que se encuentra ordenado el archivo, aunque el archivo no posea una organización especial.

A continuación se exponen algunos métodos de ordenación y búsqueda. Los algoritmos descritos pueden aplicarse tanto a archivos como a arrays o a tablas.

15.4.1 Métodos de ordenación

Ordenar consiste en colocar una serie de objetos, en nuestro caso registros de un archivo, en un determinado orden. El orden está especificado por el valor de uno o varios campos, que se denominan **llave o llaves de ordenación**. Para ordenar un registro es necesario tener acceso directo a los registros. No obstante si la organización del archivo no lo permite, se puede crear un archivo intermedio con organización aleatoria y direccionamiento directo, para después crear un archivo ordenado con la misma estructura que el de partida.

En algunos casos puede ser conveniente no aplicar el procedimiento de ordenación directamente al archivo, ya que los accesos a éste son lentos. En estos casos se creará una estructura de datos en memoria (una tabla índice) con las llaves del archivo y la dirección del registro donde se encuentran, se ordenará ésta, y posteriormente se creará el archivo de salida ordenado, tomando los datos de la tabla índice ordenada y del archivo de partida.

Hay gran cantidad de métodos de ordenación. No se pretende hacer una valoración de todos sino mostrar algunos ejemplos concretos.

Inserción

Consiste en construir una **lista** ordenada con los valores de las llaves de los registros del archivo. Una vez construida ésta, se lee secuencialmente y se escriben los registros correspondientes en el archivo de salida. Para conseguir una lista ordenada cada vez que se añade una nueva llave a ella se coloca en el lugar que le corresponda, y por tanto ésta quedará ordenada una vez que se hayan añadido a ella las llaves de todos los registros.

Mezcla

El método de mezcla (*merge*) consiste en sustituir el problema de ordenación por un problema de mezcla reiterada de secuencias del archivo ya ordenadas. Sea N el número de valores de llave distintos del archivo. En principio se parte de N segmentos, de un registro cada uno, desordenados. Estos segmentos se mezclan dos a dos produciendo N/2 segmentos ordenados de 2 registros. Este proceso se repite reiteradas veces hasta ordenar el archivo completo.

15.4.2 Métodos de búsqueda

Aquí se tratará de algunos procedimientos de localización de registros en un archivo ordenado a partir del valor de la llave, en el supuesto de que la organización del archivo sea tal que permita un acceso directo al registro sabiendo su número de orden en el archivo, y que la estructura del archivo de por sí no permite realizar una búsqueda por llave.

Indudablemente un método de búsqueda es el secuencial, del que ya se ha hablado. En dicho método hay que leer por término medio la mitad de los registros del archivo para encontrar el registro deseado. Se exponen a continuación otros métodos alternativos, como son los métodos dicotómico y de búsqueda por bloques.

Búsqueda dicotómica

Primero se lee el registro que está en el centro del archivo. Si la llave buscada es menor que la del registro leído se desecha la segunda mitad del archivo y se sigue considerando la primera; si es al revés (la llave buscada es mayor que la leída) se desecha la primera mitad del archivo y se continúa analizando la segunda mitad. A continuación se lee el registro central del tramo no desechado, y se repite el proceso anterior desechando una mitad del nuevo tramo. El proceso se continúa iterativamente hasta encontrar el registro o hasta obtener un tramo vacío (sin registros).

EJEMPLO 15.6

Para localizar el registro con llave "Pinsapo" en el archivo de la Figura 15.13 haciendo una búsqueda dicotómica los tramos de búsqueda y los registros leídos serían los que se muestran en la Tabla 15.3.

Se demuestra que el número medio de lecturas para localizar un registro en un archivo de N registros (con N grande) es:

$$\text{Log}_2(\text{N}) - 1$$

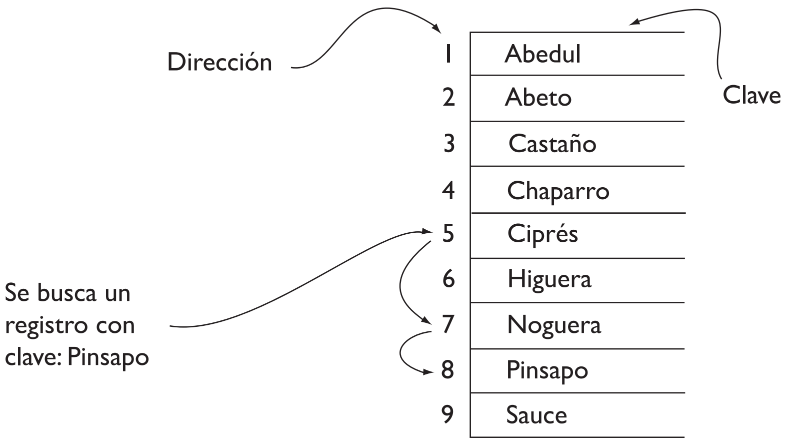


Figura 15.13. *Búsqueda por dicotomía.*

Tabla 15.3. *Rangos de búsqueda al localizar el registro Pinsapo en el archivo de la Figura 15.13.*

Iteracción	1ª	2ª	3ª
Mínimo posible	1	6	8
Registro leído	5	7	8
Máximo posible	9	9	9
Comprobación realizada	Pinsapo > Ciprés	Pinsapo > Noguera	Pinsapo = Pinsapo

Búsqueda por bloques

Con este método de búsqueda se realiza el mismo recorrido sobre las llaves que se realizaría en un archivo secuencial indexado. Para localizar un registro se considera al archivo lógicamente dividido en bloques. En primer lugar se determinará el bloque en que se encuentra el registro. Para ello se lee el último registro de cada bloque hasta, o bien encontrar el registro deseado, o bien encontrar uno mayor que él. En este último caso se procederá a localizar el registro en el bloque anterior. Esto se consigue con una búsqueda secuencial en el bloque.

15.5 Conclusión

En este capítulo se han planteado las distintas formas y soluciones en el almacenamiento y tratamiento de información en soporte permanente, externa a los programas. Los datos externos a los programas tradicionalmente se organizan en archivos. De estos se han abordado distintas organizaciones. Se han mostrado criterios, que en función de la utilización que se va a realizar del archivo, permiten decidir la organización óptima del mismo. Por último se han abordado los problemas de ordenación y búsqueda.

EJERCICIOS

15.1 Se desea almacenar un archivo de mil registros, de longitud fija, de cien caracteres cada uno en un disco. Los bloques del disco son de 512 bytes. ¿Cuál es el tiempo medio de localización de un registro para cada una de las organizaciones?. ¿Cuál será el volumen de este, si cada puntero ocupa dos bytes, y la clave de búsqueda es de diez caracteres?. ¿Que pasa si el archivo tiene diez mil registros?.

15.2 Deducir la expresión del tamaño óptimo de la zona de índices y del número de registros por tramos, de un archivo secuencial indexado.

15.3 Se desea almacenar la información que aparece en forma de esquema de base de datos, en la Figura 16.5, en un conjunto de archivos. Indicar qué archivos será necesario utilizar, indicando qué contenido tendrá cada uno y cuál debe ser su organización. ¿De qué tipo será cada uno de estos archivos?.

15.4 Escribir un procedimiento para transformar una clave alfanumérica en una dirección, para cada uno de los métodos expuestos en este capítulo.

15.5 ¿Cuantos registros se deben leer para efectuar una búsqueda por bloques?. ¿Bajo que condiciones es más rápido realizar una búsqueda por bloques que una búsqueda dicotómica? (Sugerencia: pensar en como incide el factor de blocaje en los accesos reales a disco).

15.6 Suponiendo que el archivo del Ejercicio 15.1 se ha construido con organización secuencial, comparar el tiempo medio de una búsqueda secuencial, dicotómica y por bloques. Realizar de nuevo los cálculos para un archivo de 50 registros.

15.7 Haz un organigrama del algoritmo de búsqueda dicotómica.

BIBLIOGRAFÍA

[Han85] Hanson, O.: *Essential of computers data files*, Pitman, 1985.

[Jou77] Jouffroy, C.; Letang, C.: *Les fichiers*, Dunod, 1977.

[Wir92] Wirth, N.: *Algoritmos + estructuras de datos = programas*, Prentice Hall, 1992.

[Wie87] Wiederhold, G.: *File Organization for Database Design*. Mcgraw-Hill, 1987.

CAPÍTULO 16

Bases de datos

Al igual que los archivos, las bases de datos son estructuras de datos externas a los programas, que están almacenados en disco.

Este capítulo aborda el estudio de las bases de datos, fundamentándolo en el concepto de relación. Se hace una descripción de los tipos más usuales de bases de datos en aplicaciones de gestión: estudiando las jerárquicas, en red y relacionales. Para concluir se estudia la estructura y funcionamiento de un sistema de gestión de bases de datos.

16.1 Bases de datos

En una aplicación convencional con archivos, éstos se diseñan de acuerdo con los programas. Esto es, una vez planteado el problema, se decide si debe haber archivos, cuantos deben ser, qué organización tendrán, qué información contendrá cada uno, qué programas actuarán sobre ellos y cómo lo harán. Esto tiene la ventaja, en principio, de que los programas son bastante eficientes, ya que la estructura de un archivo está pensada "para el programa" que lo va a usar. Sin embargo, conlleva graves inconvenientes. Por un lado los programas que se realizan con posterioridad a la creación de un archivo pueden ser muy lentos, al tener que usar una organización pensada y creada "a la medida" de otro programa previo que realiza procesos diferentes. Por otra parte, si se toma la decisión de crear nuevos archivos para los programas que se han de realizar, se puede entrar en un proceso de degeneración de la aplicación, ya que:

- Gran parte de *la información aparecerá duplicada* en más de un archivo (redundancias) ocupando la aplicación más espacio del necesario.
- Al existir la misma información en varios archivos, *los procesos de actualización de complican*.
- Se corre el *riesgo de tener datos incongruentes* entre los distintos archivos. Por ejemplo, tener dos domicilios diferentes de un mismo individuo en dos archivos distintos (por estar uno actualizado y el otro no).

EJEMPLO 16.1

La dirección de una empresa se plantea la necesidad de contar con información de sus empleados a efectos de envío de circulares y de selección de personal para determinadas tareas. Una solución aceptable para tal utilización puede ser crear un archivo con los siguientes campos: DNI, Nombre, Dirección, Puesto ocupado, Fecha de ingreso, Fecha de nacimiento, Teléfono, a partir de los cuales se puede obtener la información deseada.

EJEMPLO 16.1 (Cont.)

Si posteriormente la misma empresa decide informatizar su nómina tendremos que optar por una de las siguientes posibilidades:

- *Crear una aplicación completamente independiente de la anterior, que dispondrá de nuevos archivos de datos. En este caso habrá datos duplicados, y con una alta posibilidad de que en algunos casos sean incoherentes, pues los procesos de modificación serán independientes.*
- *Modificar los archivos existentes. Esto implica reescribir, o al menos modificar los programas de aplicación anteriores.*
- *Crear archivos que contengan la información de la nueva aplicación no usada en la anterior. En estos archivos deberá haber al menos un campo de enlace con el archivo anterior, por ejemplo el DNI. Nos encontramos aquí en una situación intermedia, hay menos redundancias, no es necesario reescribir todos los programas, pero los nuevos programas serán más lentos.*

En una aplicación convencional con archivos aparecen, pues, los siguientes problemas.

Dificultad de mantenimiento
Si hay archivos con información parcialmente duplicada, realizar las actualizaciones necesarias es un problema complejo y costoso. Normalmente es necesario actualizar varios archivos con diferentes organizaciones. Si la actualización no se realiza correctamente se tendrán informaciones incoherentes.

Redundancia
Se dice que hay redundancia cuando hay datos que no aportan información. Es decir, su valor se puede deducir del de otros datos.

EJEMPLO 16.2

Supóngase que en el archivo de selección del Ejemplo 16.1 se hubiese incluido un campo con la antigüedad del empleado. Este campo sería redundante, pues su valor se puede obtener del valor del campo Fecha de ingreso y de la fecha actual.

Un caso trivial de redundancia se da cuando se incluye más de una vez un mismo dato. Esto no es normal que ocurra en un archivo, pero es frecuente en un sistema de archivos.

Rigidez de búsqueda
El archivo se concibe para acceder a los datos de un determinado modo. Sin embargo, en la mayoría de los casos es necesario (o al menos deseable) combinar acceso secuencial y directo por varias claves.

Dependencia con los programas
En un archivo las relaciones existentes entre campos y registros no están plasmadas en modo alguno. Es el programa, que trabaja con el archivo, el que determina en cada caso dichas relaciones. El programa recibe cada vez que lee una cadena de caracteres. Por tanto la información

de dónde comienza y dónde acaba cada campo, su tipo, etc. está controlada y es inherente al programa. Esto implica que cualquier modificación de la estructura de un archivo obliga a modificar todos los programas que lo usen. Esto ocurre aún en el caso de que la alteración sea ajena al programa. Así si se aumenta la longitud de un campo habrá que modificar incluso los programas que no lo usan.

EJEMPLO 16.3

En el caso del archivo de dirección (Ejemplo 16.1) la organización más apropiada sería directa o indexada, utilizando como llave el DNI del empleado. Esto permite la consulta y la modificación de un empleado concreto, operaciones que serán frecuentes. Sin embargo, estas organizaciones no permiten:

- *Localizar a un empleado por su nombre.*
- *Obtener una relación de empleados por orden alfabético.*
- *Obtener empleados con fecha de ingreso anterior a una dada.*

Todas estas búsquedas implican procesar el archivo secuencialmente, creado un archivo intermedio, que en algunos casos habrá que ordenar posteriormente.

Confidencialidad y seguridad
Uno de los mayores problemas de cualquier sistema de información es mantener la reserva necesaria sobre los datos que contiene. Esto es, por un lado impedir la consulta de determinados usuarios a determinados datos (**confidencialidad**), y por otro impedir modificaciones provocadas por usuarios no autorizados (**seguridad**). Si se está trabajando con archivos el control deberá realizarlo el propio programa. En cualquier caso no se podrá impedir que alguien construya un programa para modificar o ver el contenido del archivo, si el sistema operativo le permite el acceso.

16.2 Concepto de base de datos

Las bases de datos surgen como alternativa a los sistemas de archivos, intentando eliminar o al menos reducir sus inconvenientes. Podemos definir una base de datos así:

Una **base de datos** *es un sistema formado por un conjunto de datos y un paquete software para gestión de dicho conjunto de datos de tal modo que: se controla el almacenamiento de datos redundantes, los datos resultan independientes de los programas que los usan, se almacenan las relaciones entre los datos junto con estos, y se puede acceder a los datos de diversas formas.*

En una base de datos se almacenan las relaciones entre datos junto a los datos. Esto, y el utilizar como unidad de almacenamiento el campo además del registro es el fundamento de la independencia con los programas de aplicación.

En cualquier base de datos se puede tolerar un cierto nivel de redundancia. Se utilizan las redundancias bien para hacer más rápido el acceso o bien para asegurar la integridad de los datos. En cualquier caso la gestión de la información redundante es interna a la base de datos.

Normalmente la definición del concepto de base de datos se realiza en forma de requerimientos u objetivos que ésta debe cumplir. Aunque la definición anterior es suficientemente precisa vamos a exponer algunos requisitos que debe cumplir un buen sistema de base de datos.

- **Acceso múltiple.** Diversos usuarios pueden acceder a la base de datos, sin que se produzcan conflictos, ni visiones incoherentes.
- **Utilización múltiple.** Cada usuario podrá tener una imagen o visión particular de la estructura de la base de datos.
- **Flexibilidad.** Se podrán usar distintos métodos de acceso, con tiempos de respuesta razonablemente pequeños.
- **Confidencialidad y seguridad.** Se controlará el acceso a los datos (a nivel de campo), impidiéndoselo a los usuarios no autorizados. Uno concreto podrá acceder a unos datos y a otros no.
- **Protección contra fallos.** Deben existir mecanismos concretos de recuperación en caso de fallo del computador.
- **Independencia física.** Se puede cambiar el soporte físico de la base de datos (modelo de discos, por ejemplo), sin que esto repercuta en la base de datos ni en los programas que la usan.
- **Independencia lógica.** Se pueden modificar los datos contenidos en la base, las relaciones existentes entre ellos o incluir nuevos datos, sin afectar a los programas que la usan.
- **Redundancia controlada.** Los datos se almacenan una sola vez.
- **Interfaz de alto nivel.** Existe una forma sencilla y cómoda de utilizar la base al menos desde un lenguaje de programación de alto nivel.
- **Interrogación directa ("query").** Existe una utilidad que permite al acceso a los datos de forma conversacional.

16.2.1 Estructura general de una base de datos

En una base de datos se almacena información de una serie de objetos o elementos. Estos objetos reciben el nombre de **entidades**. Entidad es cualquier ente sobre el que se almacena información. Así en una base de datos académicos podrá haber información de las siguientes entidades: alumno, profesor, asignatura, centro, plan de estudios, curso, etc. En una base de datos comerciales de una empresa podrán aparecer las siguientes entidades: cliente, producto, vendedor, etc.

De cada entidad se almacenan una serie de datos que se denominan **atributos** de la entidad. Puede ser atributo de una entidad cualquier característica o propiedad de ésta. Así son atributos de la entidad alumno: D.N.I., apellidos y nombre, sexo, fecha de nacimiento, nacionalidad, etc.

Entidades y atributos son conceptos abstractos. En una base de datos la información de cada entidad se almacena en **registros**, y cada atributo en **campos** de dicho registro, de forma análoga al almacenamiento en archivos (Secciones 1.4.2 y 15.1). Sin embargo, cada entidad necesita registros con una estructura específica, mientras que en un archivo todos los registros tienen la misma estructura. Esto es, en una base de datos hay **diferentes tipos de registros**, uno por entidad. Normalmente se reserva el nombre "registro" para especificar un "tipo de registro", usándose otras denominaciones para especificar cada una de las apariciones de ese registro en la base de datos, tales como: **elemento, valor actual de registro, valor en curso, u ocurrencia de registro**. Con esta terminología se puede decir que en la base de datos comerciales antes mencionada, hay un registro de vendedor y tantas "ocurrencias" de dicho registro como vendedores tenga la empresa.

Normalmente no es necesario conocer los valores de todos los atributos de una entidad para determinar si dos elementos son iguales. Por lo general, es suficiente con conocer el valor de uno o varios atributos para identificar un elemento. Pues bien, diremos que un conjunto de atributos de una entidad es un **identificador** de dicha entidad si el valor de dichos atributos determina de forma unívoca cada uno de los elementos de dicha entidad, y no existe ningún subconjunto de él, que sea identificador de la entidad.

EJEMPLO 16.4

En la entidad alumno el atributo DNI es un identificador. También es identificador de esta entidad el conjunto de los atributos: Apellidos y Nombre, fecha de nacimiento y lugar de nacimiento.

Frecuentemente es necesario buscar una ocurrencia de un registro en una base de datos conociendo el valor de uno o varios campos. Para que esta operación sea rápida, estos campos deben estar definidos en la base de datos como **llave** de búsqueda de dicho registro. En general podemos decir que una **llave** es un campo, o conjunto de campos, cuyos valores permiten localizar de forma rápida ocurrencias de un registro. La llave puede corresponderse con un identificador de la entidad. Si esto no ocurre podrá haber varias ocurrencias de registro con el mismo valor de llave. Se dice entonces que la llave admite duplicados.

En una base de datos se almacenan además de las entidades las **relaciones** existentes entre ellas. Así, por ejemplo, en la base de datos académicos antes citada hay relaciones entre las siguientes entidades: cursos y alumnos, alumnos y profesores, profesores y asignaturas. Estas relaciones se explicitan con punteros que inserta automáticamente el sistema de gestión de la base de datos; es decir, los punteros son “transparentes” al usuario.

Las relaciones entre entidades pueden ser simples o complejas (Figura 16.1). Una relación es **simple** cuando es biunívoca (uno a uno). Es simple la relación existente entre alumno y expediente o profesor y horario. Cualquier relación que no sea biunívoca se dice que es **compleja**. Son complejas las relaciones de uno a muchos, la existente entre un alumno y las asignaturas de las que está matriculado. Las relaciones muchos a muchos son también complejas. A este último grupo pertenece la relación existente entre alumno y profesor. Al representar gráficamente la estructura de una base de datos, los registros se simbolizan mediante cajas y las relaciones mediante fechas.

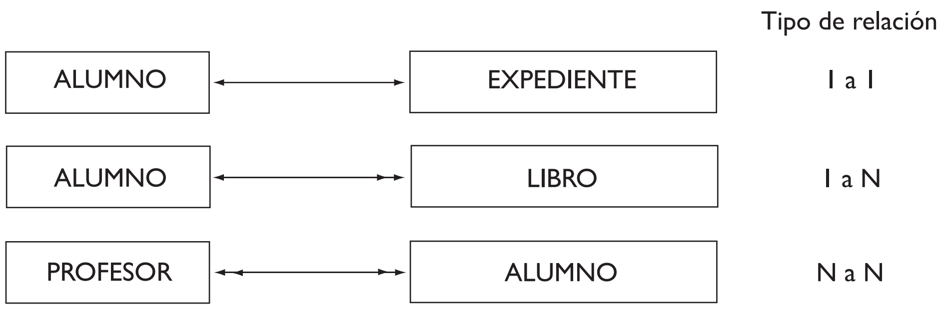


Figura 16.1. *Representación de los distintos tipos de relaciones.*

No se deben confundir las relaciones con simples referencias. Decimos que existe una **referencia** entre dos registros cuando un campo de uno de ellos es llave en el otro. Esto permite localizar uno de ellos a partir del otro. Sin embargo, no se trata de una relación. La relación está ligada a la estructura de la base de datos no al valor de los campos almacenados.

EJEMPLO 16.5

Supóngase que la base de datos de personal de una empresa tiene las entidades Empleados y Puestos, con la estructura que se muestra en la Figura 16.2. Ambas entidades están ligadas por el atributo Puesto. A partir del valor del atributo "puesto de un empleado" se puede encontrar la ocurrencia asociada del registro de "puestos". Entre ambos registros hay pues una **referencia***, que como se ve, implica la existencia de un campo redundante. No obstante es mejor ligar ambos registros con una* **relación** *(Figura 16.3) ya que ésta no implica redundancia debido a que el campo Puesto no aparecerá en el registro de empleados. Hay un puntero interno a la dirección del elemento de la entidad "puesto" correspondiente.*

EMPLEADOS

Apellidos y nombre	DNI	Puesto	Antigüedad

PUESTOS

Puesto	Sueldo	Base de cotización

Figura 16.2. *Registros de la base de datos del Ejemplo 16.5, con referencia entre ambos.*

El usar referencias entre diferentes registros implica incluir redundancias en la base de datos. Por otra parte, una base de datos con redundancias es difícil de mantener. Piénsese en la estructura de la Figura 16.2. Si se necesita cambiar el nombre de un puesto será necesario recorrer todas las ocurrencias del registro empleados, modificando el valor del campo puesto. El uso de referencias debe pues evitarse, ya que atenta contra los propios objetivos de las bases de datos.

Al definir la base de datos se deben especificar cada uno de los registros que la integran, indicando los campos que lo componen, y las relaciones que los ligan. Tal conjunto de especificaciones recibe el nombre de esquema de la base de datos. El **esquema de una base de datos** *es pues la definición de su estructura lógica. El esquema suele representarse en forma gráfica (Figura 16.4) o por simple enumeración.*

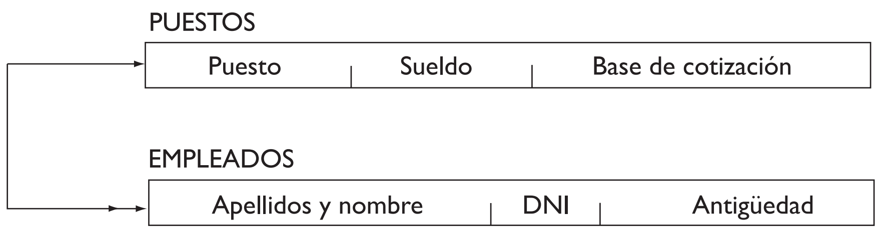


Figura 16.3. *Estructura de la base de datos del Ejemplo 16.5, sin referencia entre ambos registros.*

Para que un programa concreto pueda acceder a una base de datos es necesario especificar la estructura lógica de la parte de base de datos que éste va a usar. La descripción de la estructura

lógica de una parte de la base de datos, que va a ser usada por uno o más programas, recibe el nombre de **subesquema**, éste está siempre contenido en el esquema. No obstante los nombres de campos, registros y relaciones pueden ser diferentes en el esquema y subesquema. El subesquema consigue aislar los programas de la estructura lógica de la base de datos. Por otra parte permite que varios usuarios utilicen distintas estructuras de una misma base de datos. El subesquema permite además limitar el acceso a distintas partes de la base de datos, para realizar tan sólo determinadas acciones.

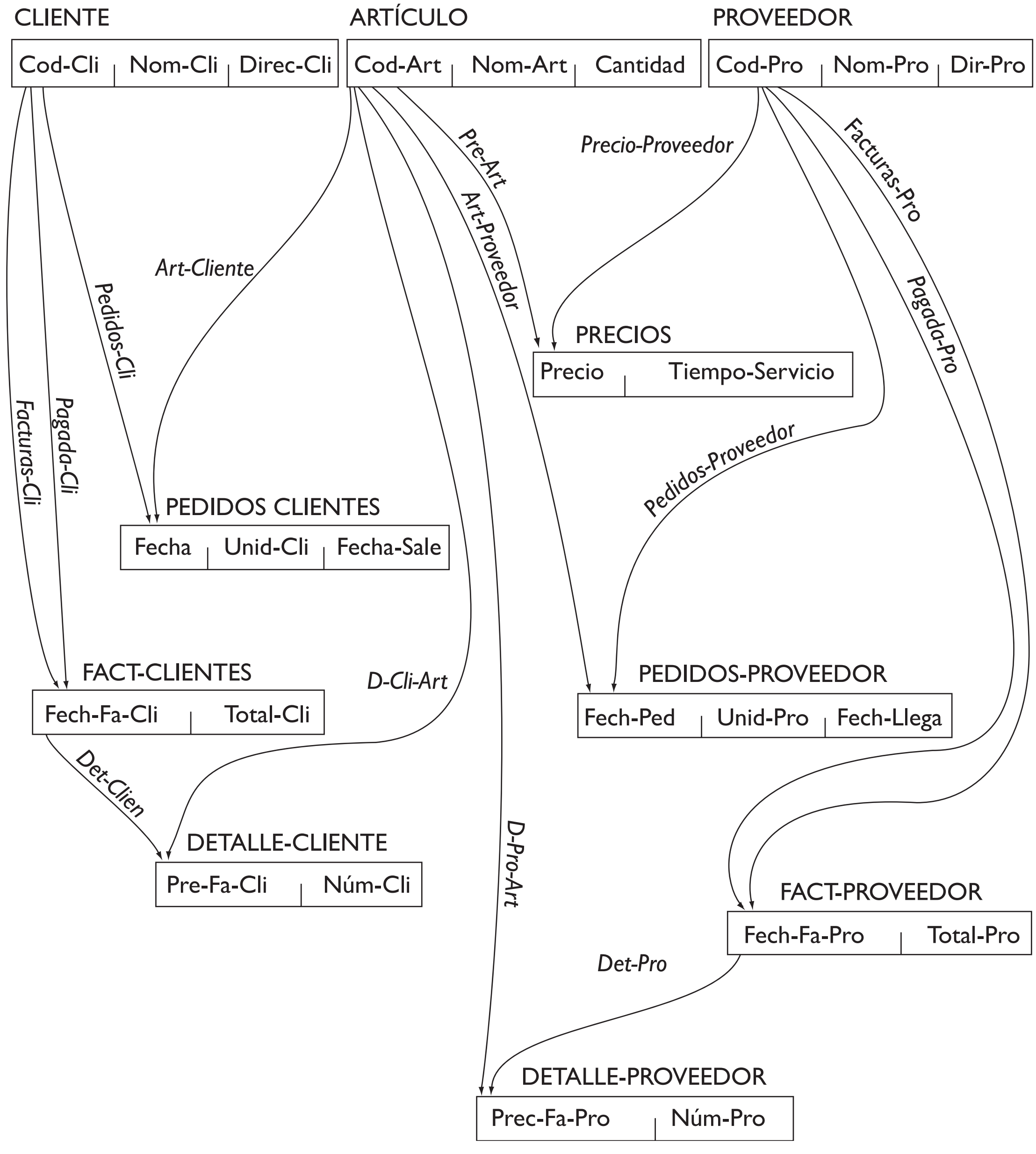


Figura 16.4. *Esquema de una base de datos para el Ejemplo 16.4.*

EJEMPLO 16.6

Supóngase que una empresa de suministros desea informatizar sus operaciones de gestión, que incluyen control de existencias, control de ventas, facturación a clientes, pedidos y facturas de proveedores. El esquema de la base de datos que se muestra en la Figura 16.4 permite realizar las operaciones normales sobre estas entidades. En este esquema se observa que:

- *Entre algunos registros hay más de una relación. Así entre CLIENTES y FACT-CLIENTES hay dos relaciones, una liga cada cliente con sus facturas pagadas y la otra liga cada cliente con las facturas pendientes de pago. Una factura concreta sólo se relaciona con un cliente por una de las dos relaciones.*
  - *Los nombres de campos, registros y relaciones son únicos en todo el esquema. Es importante que el nombre de cada elemento exprese lo que contiene (sea significativo). En cada subesquema se podrán acortar los nombres, si ello no induce a error.*
- *No aparecen campos duplicados, no hay pues referencias.*
- *Hay registros a los que se accede directamente. Así a los registros CLIENTE, ARTÍCULO y PROVEEDOR se accede sin especificar previamente ningún otro registro.*

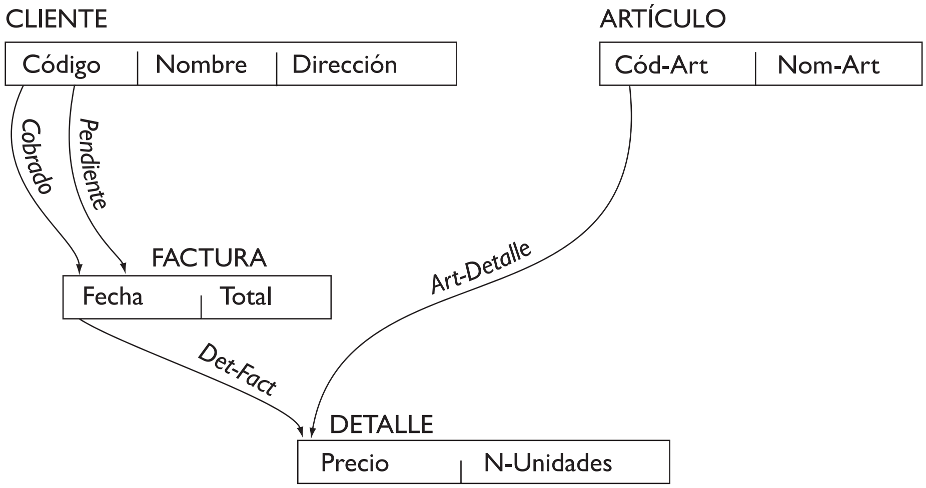


Figura 16.5. *Subesquema para los programas de gestión de cobros.*

Para cada programa que acceda a la base de datos será necesario definir un subesquema. Para los programas que se encarguen de la gestión de cobro a clientes puede definirse el subesquema de la Figura 16.5. Este subesquema es, como puede apreciarse, una parte del esquema de la Figura 16.4. En él se han modificando los nombres de algunos elementos.

16.3 Tipos de bases de datos

Las bases de datos se clasifican tradicionalmente en tres grupos: *jerárquicas*, *en red* y *relacionales*. Las dos primeras se diferencian en los tipos de relaciones que permiten. Puede

decirse que la estructura jerárquica es un caso particular de la estructura en red. Cualquier esquema que se cree para una base de datos jerárquica se puede utilizar para una base de datos en red. Las bases de datos relacionales son conceptualmente distintas. En estas, las relaciones se almacenan y manipulan como tablas, como se verá más adelante (Sección 16.3.3).

16.3.1 Bases de datos jerárquicas

En una base de datos jerárquica sólo se pueden crear estructuras jerárquicas (esto es, en árbol). No es pues posible definir relaciones muchos a muchos. Una base de datos jerárquica tiene una estructura semejante a la de la Figura 16.6.

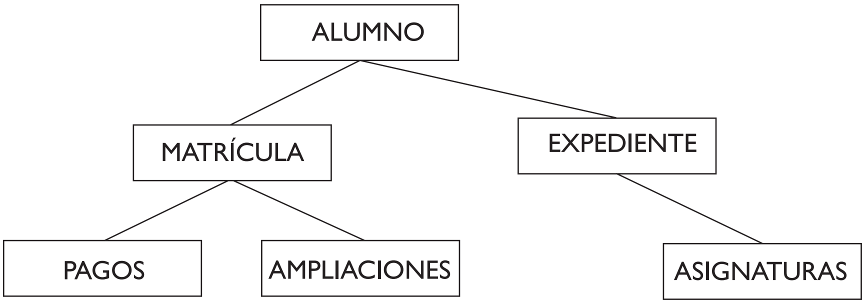


Figura 16.6. *Esquema de una base de datos jerárquica.*

Para poder dar una caracterización más precisa de este tipo de base de datos se introduce un nuevo concepto: el de conjunto. Un **conjunto** *es una estructura formada por dos registros ligados por una relación uno a muchos*. Los registros que forman el conjunto reciben, en éste, los nombres de **propietario** y **miembro**, siendo la relación de un propietario a muchos miembros (Figura 16.7).

Se dice que una base de datos es **jerárquica** cuando no permite definir relaciones muchos a muchos y no acepta estructuras en las que un mismo registro sea miembro de dos conjuntos distintos.



Figura 16.7. *Conjunto.*

Físicamente una estructura de este tipo se almacena usando punteros como enlace entre los distintos registros de cada conjunto, como se muestra en la Figura 16.8. Básicamente hay dos posibles realizaciones:

- Usando sólo punteros propietario-miembro y miembro-propietario. Presenta el inconveniente de que el número de punteros usados en cada registro es variable y puede ser grande.
- Usando punteros propietario-primer miembro y punteros entre miembros.

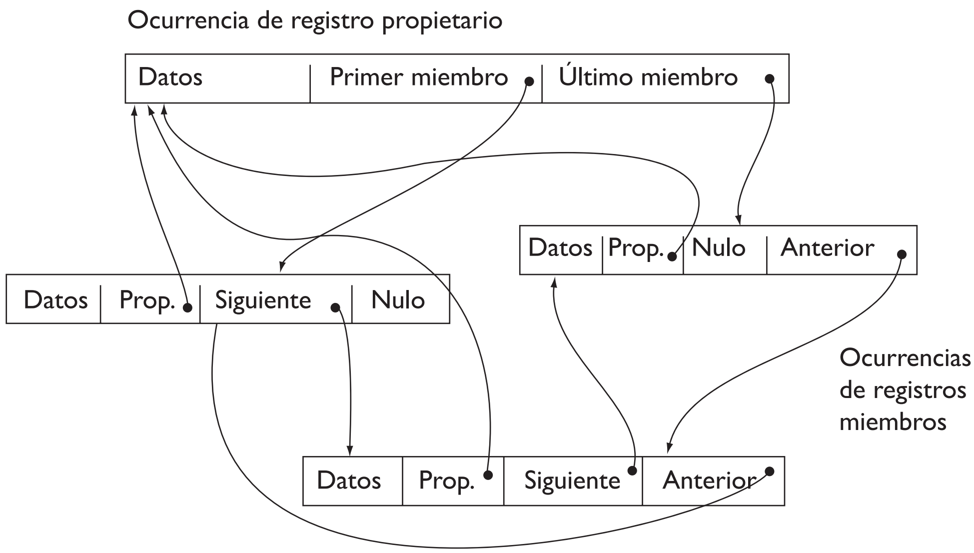


Figura 16.8. *Representación física de un conjunto.*

16.3.2 Bases de datos en red

En una base de datos en red no hay ninguna restricción ni en el tipo de relaciones que se pueden usar, ni en los registros que pueden intervenir en ellas. No obstante, se distinguen entre bases de datos en red **simple** y bases de datos en red **compleja**, según permitan o no utilizar relaciones muchos a muchos. En una base de datos en red simple este último tipo de relaciones no está permitido. Éste es, por otra parte, el tipo de sistema de base de datos más usual.

Una base de datos en red simple se puede descomponer en conjuntos, al igual que una base jerárquica.

La limitación en el tipo de relaciones que permite tratar no implica restricción alguna en el tipo de datos que se pueden almacenar. Cualquier conjunto de datos es representable en cualquier sistema de base de datos en red simple. Tomemos por ejemplo la relación profesor-alumno. Ésta es una relación muchos a muchos. Dicha relación se puede incluir en una base de datos en red compleja, pero, en principio, no en una base en red simple. Esta información, para ser almacenada en una base de datos en red simple (abreviadamente “en red”), debe expresarse sustituyendo la relación muchos a muchos por dos relaciones uno a muchos (véase Figura 16.9). Para ello se introduce un nuevo registro del que existirá una ocurrencia por cada par profesor-alumno que estén relacionados, y que llamaremos AL-PRO. De alguna forma se puede decir que el registro profesor está relacionado con AL-PRO, mediante una relación uno a muchos. Del mismo modo existe una relación uno a muchos entre alumnos y AL-PRO. La estructura generada es ahora representable en una base de datos en red simple. Además da la misma información que la estructura de partida.

El registro AL-PRO es en cierto modo artificial, aunque puede ser útil para almacenar información propia del par profesor-alumno. Por ejemplo, fecha de contactos o calificaciones. A los registros introducidos en una estructura para evitar la utilización de relaciones muchos a muchos se les denomina **enlaces** (*link*). La inclusión de un enlace no implica introducir redundancias en la base de datos.

Para almacenar la misma información en una base de datos jerárquica es necesario sustituir alguna relación por una referencia.

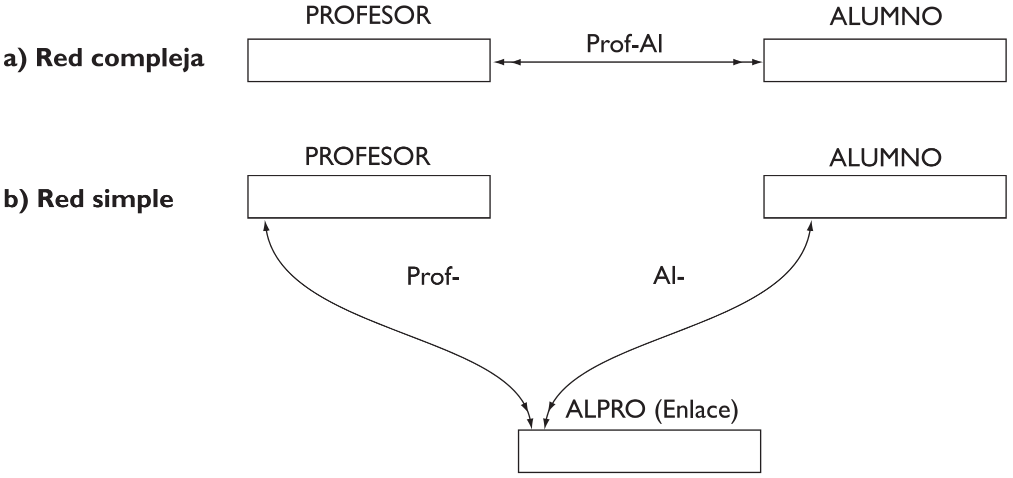


Figura 16.9. *Representación de la relación Alumno-Profesor en bases de datos en red compleja y simple.*

16.3.3 Bases de datos relacionales

A principios de la década de los 70 E. F. Codd planteó una alternativa a las bases de datos jerárquicas y en red vistas hasta aquí. Pretendía obtener más flexibilidad que con las bases anteriores y más rigor en el tratamiento de los datos.

Una base de datos relacional está formada por tablas. Una tabla es una estructura bidimensional formada por una sucesión de registros del mismo tipo.

Número de vuelo	Avión	Origen	Destino	Hora de salida
27	DC-9	Granada	Madrid	8:45
404	DC-10	Madrid	Oslo	10:37
1024	B-727	Barcelona	París	9:45
114	B-727	Santiago	Sevilla	17:21
503	DC-9	Madrid	Málaga	15:30

Figura 16.10. *Representación de una tabla.*

Si se imponen ciertas condiciones a las tablas, se pueden tratar como **relaciones** matemáticas. De ahí el nombre de este tipo de bases de datos y el hecho de que a las tablas de una base de datos relacional se les denomine **tablas relaciones**. Las tablas deben cumplir las siguientes condiciones:

- Todos los registros de una tabla son del mismo tipo. Para almacenar registros de tipos distintos se usan tablas distintas.
- En ninguna tabla aparecen campos repetidos.
- En ninguna tabla existen registros duplicados.
- El orden de los registros en la tabla es indiferente. En cada momento se pueden recuperar los registros en un orden particular.
- En cada tabla hay una llave, formada por uno o varios campos.

EJEMPLO 16.7

La estructura en red del ejemplo de la Figura 16.5 requiere las cinco tablas que se muestran en la Figura 16.11, para ser representada en una base de datos relacional.

PENDIENTES DE COBRO

N.º Factura	Cod-Cli	Fecha	Total
10234	231	3/4/6	12356
		1/2/6	

CLIENTES

Cod-Cli	Nom-Cli	Dirección
231	Pablo	Cuesta, 4
		Jardín, 3

ARTÍCULOS

Cod-Art	Nom-Art
1001	Alón
1003	

COBRADOS

N.º Factura	Cod-Cli	Fecha	Total
13	231	4/5/6	9733
		12/8/6	

DETALLE FACTURA

Cod-Cli	Núm-Fac	Cod-Art	Precio	N.º unidades
231	13	1003	124	26
	13	1001	320	

Figura 16.11. *Ejemplo de tablas de una base de datos relacional.*

Comparando la estructura relacional con la estructura en red, se observa que ha sido necesario incluir campos adicionales en algunos registros para disponer de una llave en cada uno, pudiéndose así relacionar la información de las distintas tablas.

16.4 Sistema de gestión de la base de datos

Se denomina sistema de gestión de la base de datos (o **DBMS**, del inglés "Data Base Management System") al conjunto de software destinado a la creación, control y manipulación de la información de una base de datos. Concretamente un DBMS debe permitir la realización de las siguientes tareas:

- **Acceso** a los datos desde algún lenguaje de alto nivel.
- **Interrogación** (o recuperación de la información) directa en modo conversacional.
- **Definición** del esquema de la base de datos y de los distintos subesquemas.
- **Organización** física de la base de datos y recuperación tras fallos del sistema.

Las tres primeras funciones se realizan mediante dos lenguajes específicos:

- **Lenguaje de manipulación de datos** (o **DML**, del inglés "Data Manipulation Language"). Se utiliza para el acceso a la base de datos desde lenguajes de alto nivel o en modo conversacional.
- **Lenguaje de Descripción de Datos** (o **DDL**, del inglés "Data Description Language"). Utilizado para la descripción del esquema y de los subesquemas.

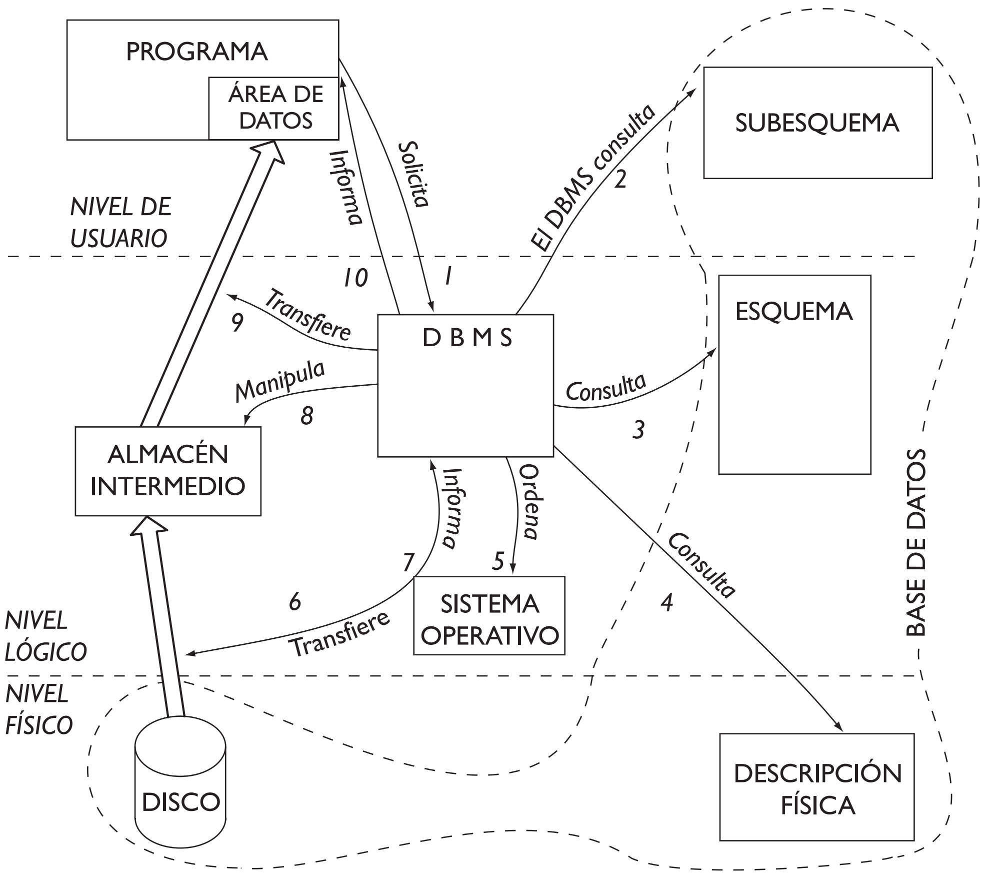


Figura 16.12. *Acciones realizadas por el DBMS en una transferencia de información de la base de datos a un programa de aplicación.*

El **DBMS** actúa como intermediario entre los programas de aplicación y el sistema operativo (véase Figura 16.12). Esto permite que los programas sean independientes de la estructura física de los datos.

A continuación se exponen con más detalle las características y modo de utilización del DDL y del DML en bases de datos jerárquicas y en red. Al estudio de la estructura de estos lenguajes para bases de datos relacionales se dedicará la Sección 16.4.3.

16.4.1 Lenguaje de descripción de los datos

El lenguaje de descripción de datos (DDL) es el lenguaje usado para definir la estructura lógica de la base de datos. Esto es, crear el esquema y los subesquemas. Por extensión se denomina DDL al software que interpreta este lenguaje.

La definición del esquema consiste en la enumeración de los registros que van a existir en la base de datos, especificando para cada uno de ellos los campos de que consta y los conjuntos de que forma parte. Para describir cada registro es necesario especificar:

- Nombre del registro.
- Descripción de campos. Consistente, para cada campo en:
  - Nombre del campo.
- Tipo (numérico, alfabético, alfanumérico, o binario).
  - Tamaño en bytes ocupado por el campo.
  - Restricciones en los valores que puede tomar.

A continuación se muestra un ejemplo concreto de descripción de registro.

EJEMPLO 16.8

La definición del registro de cliente de la base de datos del Ejemplo 16.6, en DDL del DG/DBMS es:

```
RECORD NAME IS CLIENTE
    02    COD-CLI      PICTURE IS X(3)
    02    NOM-CLI      PICTURE IS X(24)
    02    DIREC-CLI    PICTURE IS X(24)
```

Al describir cada conjunto se debe especificar:

- *Nombre del conjunto.*
- *Nombre del registro propietario.*
- *Nombre del registro miembro.*
- *Criterio de ordenación entre miembros.*
- *Modo de retención.*
- *Modo de inserción.*

En cada conjunto se pueden recuperar todos los registros miembros relacionados con un propietario determinado. El orden en que estos se obtienen depende del **criterio de ordenación**

utilizado al insertar nuevos miembros en el conjunto. Los criterios de ordenación más frecuentes son:

- **FIRST.** Cada nuevo registro pasa a ser el primero de la cadena de registros miembros. Será pues el primero en leerse. En este caso la cadena de miembros es una pila.
- **LAST**. Cada nuevo miembro se coloca al final. De este modo los registros miembros forman una cola.
- **SORT**. Los miembros se almacenan ordenados según el valor de una llave. Esto permite localizar un miembro sabiendo el valor de su llave y obtener los miembros ordenados según el valor de llave.

EJEMPLO 16.9

La definición del conjunto DET-CLIEN de la base de datos del Ejemplo 16.6 para DDL de DG/DBMS es:

```
SET NAME IS DET-CLI
    OWNER IS FACT-CLIENTES
    MEMBER IS DETALLE-CLIENTE
        INSERTION IS AUTOMATIC
        RETENTION IS MANDATORY
        MEMBER LIMIT IS 50
        ORDER IS LAST
```

El modo de inserción determina la acción del sistema al incluir una ocurrencia del registro miembro en el conjunto. Si el modo de inserción es automático se le asigna como propietario la última ocurrencia de registro propietario a la que se ha accedido. Si el modo de inserción es manual la ocurrencia de registro introducida no se conecta a ningún propietario. La conexión deberá realizarse expresamente.

En algunos casos será deseable poder desconectar una ocurrencia de registro miembro de su propietario; por ejemplo, para cambiar el propietario. En este caso el modo debe especificar como modo de retención opcional (o *manual*). Por el contrario, si se desea impedir que una ocurrencia de registro miembro se desenganche de su propietario, se deberá usar el modo de retención mandatorio (o *automático*).

EJEMPLO 16.10

Supongamos que la base de datos académicos de un centro docente (en que cada curso es selectivo) tiene la estructura que se muestra en la Figura 6.13.

Dado que un alumno puede cambiar de curso, el conjunto AL-CUR se debe definir con modo de retención opcional. Por otra parte es posible que al comenzar un curso no se sepa qué profesor va a impartir cada asignatura, por tanto el conjunto PRO-AS debe definirse con modo de inserción manual. Una asignatura se define para un curso y no se cambia de curso, el conjunto C-AS se puede definir con modo de inserción automático y modo de retención mandatorio.

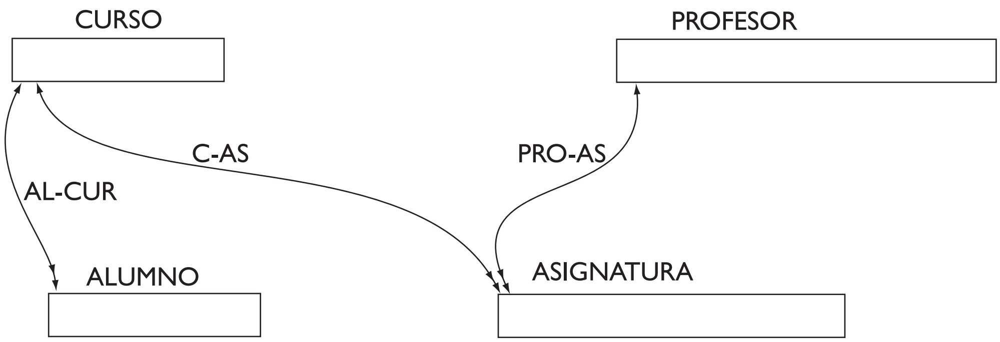


Figura 16.13. *Estructura de la base de datos de un centro docente.*

EJEMPLO 16.11

Se desea obtener una lista de asignaturas con profesores que las imparten, ordenadas por cursos, de la base de datos del Ejemplo 16.9. Para ello se puede usar el siguiente procedimiento:

```
find primer CURSO
get CURSO
mientras no error
    escribe CURSO
    find ASIGNATURA en conjunto C-AS
    get ASIGNATURA
    mientras no error
        find propietario en conjunto PRO-AS
        get PROFESOR
        si no error
            escribe ASIGNATURA, PROFESOR
        si no
            escribe ASIGNATURA, "sin asignar"
        fin si
        find ASIGNATURA en conjunto C-AS
            get ASIGNATURA
        fin mientras
        find CURSO
        get CURSO
    fin mientras
```

16.4.2 Lenguaje de manipulación de los datos

El lenguaje de manipulación de los datos (DML) es el lenguaje usado para acceder a la base de datos, para leer, escribir o modificar información. Las sentencias de DML se pueden utilizar de forma interactiva o bien desde algún lenguaje de alto nivel. Cuando se utilizan

sentencias DML desde un lenguaje de alto nivel se debe preparar el texto del programa para que el compilador pueda traducirlo. En algunos casos el compilador entiende directamente las sentencias de DML, se dice en este caso que el DML está alojado en el compilador. Normalmente esto no sucede y se debe realizar un paso previo antes de compilar el programa; esto es, se tiene que utilizar un **precompilador**. El precompilador se encarga de sustituir las sentencias DML por llamadas a rutinas.

El repertorio de sentencias del DML es variable según el sistema de bases de datos usado. Debe incluir al menos las siguientes sentencias:

- **FIND.** Busca una ocurrencia de registro miembro que cumpla una determinada condición, en un conjunto dado y conectada a un propietario concreto. O bien busca el propietario de un registro dado.
- **GET.** Recupera, o lee, una ocurrencia de registro.
- **STORE.** Almacena, o escribe, una ocurrencia de registro.
- **MODIFY.** Modifica una ocurrencia de registro.
- **ERASE.** Borra una ocurrencia de registro.
- **VCONNECT.** Conecta una ocurrencia de registro miembro a un propietario.
- **DISCONNECT.** Desconecta un miembro de su propietario.

16.4.3 Sistemas de gestión de bases de datos relacionales

Lo dicho anteriormente se refiere exclusivamente a bases de datos jerárquicas y en red. Tanto el DDL como el DML de un sistema de base de datos relacional son diferentes a los descritos en los apartados anteriores. Nos ocuparemos brevemente aquí de la descripción de ambos lenguajes para bases de datos relacionales.

Para definir una base de datos relacional se deben especificar cada una de las tablas que la constituyen. La descripción de una tabla consta de:

- Nombre de la tabla.
- Descripción de los campos de que consta cada registro de la tabla. La descripción de los campos es semejante a la realizada para sistemas en red.
- Nombre de los campos que se usan como llave.

A continuación se incluye un ejemplo de descripción de una tabla.

Las sentencias del DML de una base de datos relacional realizan operaciones con las tablas. Algunas de estas operaciones son: proyectar, seleccionar y componer o unir (véase Figura 16.14).

La **proyección** de una tabla es otra tabla que contiene los mismos registros que la tabla de partida, y sólo determinadas columnas. Al proyectar una tabla se especifican las columnas que se desean obtener.

Una **selección** en una tabla produce como resultado otra tabla que contiene los registros que cumplen una determinada condición. La condición se expresa en función de los valores de los campos. La tabla creada al seleccionar contiene todas las columnas de la tabla original.

La **composición** o **unión** es una operación que se realiza con dos tablas, para producir una nueva tabla. La unión se realiza en base al valor de algún campo. Para cada par de registros de las tablas a componer, que tengan el mismo valor en dicho campo, se añade un registro en la tabla de salida. La tabla de salida contiene los campos que se encuentran en alguna de las tablas de entrada.

AUTORES

CÓDIGO	AUTOR
001	Juan Pérez
001	Antonio Sánchez
002	Luis Espina
003	Ana García
004	José Tilo
004	Antonio Cantor
004	Pedro Canijo
005	Francisco Pelo

TÍTULOS

CÓDIGO	TEMA	TÍTULO
001	Datos	Escructuras de datos
002	Micros	Microprocesadores
003	Lengua	Programación en Cobol
004	Datos	Formaciones en Pascal
005	Lengua	Basic compilado

Proyección de TÍTULOS sobre TEMA Y TÍTULO

TEMA	TÍTULO
Datos	Escructuras de datos
Micros	Microprocesadores
Lengua	Programación en Cobol
Datos	Formaciones en Pascal
Lengua	Basic compilado

Selección en TÍTULOS por TEMA = «Lengua»

CÓDIGO	TEMA	TÍTULO
003	Lengua	Programación en Cobol
005	Lengua	Basic compilado

Unión de AUTORES y TÍTULOS por CÓDIGO

CÓDIGO	TEMA	TÍTULO	AUTOR
001	Datos	Escructuras de datos	Juan Pérez
001	Datos	Escructuras de datos	Antonio Sánchez
002	Micros	Microprocesadores	Luis Espinar
003	Lengua	Programación en Cobol	Ana García
004	Datos	Formaciones en Pascal	José Tilo
004	Datos	Formaciones en Pascal	Antonio Cantor
004	Datos	Formaciones en Pascal	Pedro Canijo
005	Lengua	Basic compilado	Francisco Pelo

Figura 16.14. *Operaciones básicas con tablas.*

La manipulación de una base de datos relacional es muy distinta a la de estructura en red. En una base de datos en red el usuario se mueve por la estructura, mientras que en una base de datos relacional realiza operaciones con tablas.

Cualquier DML relacional debe tener sentencias que realicen estas operaciones, y sentencias que permitan actualizar la información contenida en las tablas. Cualquier DML deberá contener al menos sentencias equivalentes a las siguientes:

- **APPEND.** Añade un registro a una tabla.
- **EDIT.** Modifica un registro de una tabla.
- **DELETE.** Borra un registro de una tabla.
- **DISPLAY.** Proyecta una tabla.
- **DISPLAY FOR.** Selecciona registros en una tabla.
- **JOIN.** Une dos tablas.

EJEMPLO 16.12

Para gestionar los datos del centro docente del Ejemplo 16.10 con una base de datos relacional, se puede usar la estructura de la Figura 16.15. Si sobre esta base de datos se desea obtener una lista de asignaturas con profesores que las imparten, tal como se hizo en el Ejemplo 16.11 para una base de datos en red, se usarán las siguientes sentencias del DML:

***join** PROFESOR **with** ASIGNATURA **for** COD-ASIG **to** PROF-AS*

***display** CURSO, NOM-ASIG, NOM-PROF*

La primera sentencia crea una tabla resultado de la unión de las tablas PROFESOR y ASIGNATURA por el campo de COD-ASIG. La segunda sentencia proyecta esta tabla sobre los campos CURSO, NOM-ASIG y NOM-PROF.

ALUMNO

Nom-Alum	Curso
Antón	2
Catón	1
Diego	

PROFESOR

Nom-Prof	Cod-Asig
Pérez	11
Ruiz	12
	13

ASIGNATURA

Cod-Asig	Curso	Nom-Asig
11	1	Informática
12	1	Programación
13		

Figura 16.15. *Base de datos de un centro docente.*

16.5 Conclusión

En este capítulo se han planteado las distintas formas y soluciones en el almacenamiento y tratamiento de información en soporte permanente, externa a los programas. Los datos externos a los programas tradicionalmente se organizan en archivos. De éstos se han abordado distintas organizaciones. Se han mostrado criterios, que en función de la utilización que se va a realizar del archivo, permiten decidir la organización óptima del mismo.

Cuando la información a procesar es compleja (datos interrelacionados de distintos objetos) es más aconsejable la utilización de bases de datos [1,4,5]. Se han discutido los tipos de bases de datos usuales en aplicaciones de gestión, y se ha hecho una descripción general del funcionamiento de un sistema de gestión de bases de datos.

EJERCICIOS

16.1 ¿Cuál debe ser el modo de inserción y eliminación de cada uno de los conjuntos de la base de datos de la Figura 16.4?.

16.2 Dibujar un esquema de base de datos en red para el hotel del Ejemplo 15.2.

16.3 Realizar la definición del esquema de la base de datos de la Figura 16.3.

16.4 Sobre la base de datos de la Figura 16.4, describir el procedimiento de escritura de una factura de un cliente.

16.5 Modificar la estructura de la base de datos del Ejemplo 16.10 para contemplar la existencia de más de un grupo por asignatura.

16.6 Indicar cómo se afectaría la modificación del ejercicio anterior a la base de datos relacional del Ejemplo 16.13.

16.7 Sobre la base de datos del ejercicio anterior, ¿qué operaciones deberían realizarse para generar un listado de alumnos por grupo?

16.8 Indicar cómo se puede generar un listado de clientes que tienen facturas pendientes de cobro.

BIBLIOGRAFÍA

[Dat01] Date, C. J.: *Introducción a los sistemas de base de datos,* Pearson, 2001.

[Kor00] Korth, H. F.; Silverschatz, A.: *Fundamentos de bases de datos,* McGraw-Hill, 2000.

[Mar80] Martin, J.: *Organización de las bases de datos,* Prentice Hall Internacional, 1980.

[May85] Mayne, A.; Wood, M. B.: *Introducción a las bases de datos relacionales,* Diaz de Santos, 1985.

[Sil06] Silberschatz, A.; Korth, H. F.; Sudarshan, S.: *Fundamentos de Bases de Datos.* 5ª Edc., McGraw-Hill, 2006.

[Ulm01] Ullman, J. D; Widom, J. D.: *A First Course in Database Systems,* Prentice Hall, 2001.

[Wie83] Wiederhold, G.: *Data Base desing,* McGraw-Hill, 1983.

CAPÍTULO 17

Ingeniería del software

Este capítulo se dedica a introducir el estudio del proceso de construcción de una aplicación informática, abordando tanto los métodos usados como las etapas a seguir, lo que globalmente se conoce como *ingeniería del software*.

Cuando uno desarrolla una aplicación para uso personal, como un "entretenimiento", el esfuerzo que deba hacer, el tiempo necesario para concluirlo, o hasta qué punto satisface nuestras expectativas, no son problemas importantes. Cuando, por el contrario, se desarrolla una aplicación para un cliente que tiene unas necesidades específicas, que necesita el producto en un momento concreto y que va a pagar por el desarrollo, el coste, el plazo de realización, y el que se satisfagan los requisitos de cliente, son esenciales.

La ingeniería del software se ocupa del estudio de los problemas relacionados con el desarrollo de software y de sus soluciones. Se puede definir la **ingeniería del software** (siguiendo a Fritz Baner) como: *La disciplina que se ocupa del establecimiento y uso de principios firmes de ingeniería, para obtener software económico que sea fiable y funcione eficientemente en máquinas reales* [Pre02].

En este capítulo se hará un especial énfasis en la planificación, y prueba de un sistema software.

17.1 El desarrollo de sistemas software

En el Capítulo 3 se ha tratado el problema del diseño y construcción de un *programa*. Este problema puede ser, en algunos casos, bastante complejo. No obstante, no es usual que la resolución de un problema real implique el desarrollo de un solo programa. El desarrollo de cualquier aplicación práctica es más difícil que el de un programa individual. Esto es así fundamentalmente por los siguientes motivos:

- **Volumen.** Un proyecto, en la práctica, suele ser grande, (decenas o cientos de miles de líneas de código).
- **Evolución.** El software no es estático. Evoluciona con las necesidades del usuario, los cambios del entorno (hardware, legislación, etc.).
- **Complejidad.** El software es complejo. Es difícil que una única persona conozca todos los detalles de una aplicación.
- **Coordinación.** En el desarrollo de una aplicación intervienen muchas personas. Es difícil coordinar el trabajo de todos, de forma que al final los distintos componentes encajen.
- **Comunicación.** Cuando se desarrolla una aplicación es porque, de una forma u otra, hay alguien interesado en usarla. Antes de comenzar el desarrollo, habrá que concretar

con el usuario (o cliente) las características de la aplicación. Esta comunicación con el cliente conlleva serios problemas, pues normalmente el informático no conoce las necesidades del cliente, y el cliente no sabe discernir la información que es útil para el informático de la que no es.

La ingeniería del software se ocupa de la planificación y estimación de proyectos, análisis de requerimientos, diseño del software, codificación, prueba y mantenimiento. Para realizar estas tareas propone una serie de métodos. La realización de un proyecto se consigue aplicando los métodos siguiendo una determinada secuencia.

Lógicamente la ingeniería del software tiene puntos en común con cualquier otra ingeniería. No obstante existen grandes diferencias con otras ingenierías. La principal diferencia procede de la ausencia de proceso de fabricación en el desarrollo de software. Cuando se construyen automóviles una vez diseñados se fabrican. La ausencia de proceso de fabricación en la ingeniería software hace que el coste de la producción sea fundamentalmente el coste del diseño.

Por otra parte, el mantenimiento del software es muy diferente al del hardware (o cualquier otro sistema físico). El software no se desgasta con el tiempo. El mantenimiento del software está motivado por la detección de fallos o por la necesidad de adaptarlo a unas nuevas circunstancias (evolución). En ambos casos, el proceso de mantenimiento no consiste en la sustitución de un componente por otro, sino en la repetición de parte del proceso de desarrollo.

17.2 Ciclo de vida del software

El desarrollo de cualquier sistema software implica la realización de tres pasos genéricos: definición, construcción y mantenimiento.

En la **fase de definición** se intenta caracterizar el sistema que se ha de construir. Esto es, determinar qué información ha de usar el sistema, qué funciones ha de realizar, qué condicionantes existen, cuáles han de ser las interfaces del sistema, y qué criterios de validación se usarán. Esto es, se debe contestar a la pregunta ¿qué hay que desarrollar?

En la **fase de construcción** se diseñan las estructuras de los datos (bases de datos o archivos) y de los programas, se escriben y documentan éstos, y se prueba el software.

La **fase de mantenimiento** comienza una vez construido el sistema, coincidiendo con su vida útil. Durante ella, el software es sometido a una serie de modificaciones y reparaciones.

En cada una de estas fases se pueden detectar problemas que obliguen a modificar algo que se hizo en una fase anterior (Figura 17.1).

La detección de errores durante la definición y la construcción se realiza mediante *revisiones* de la documentación y de los productos generados en cada fase. En estas revisiones se pueden detectar fallos o inconsistencias que obliguen a repetir parte de la fase, o una fase anterior.

El conjunto de información generada en todas las fases constituye la **configuración del software**. *Un buen sistema software no es solamente un conjunto de programas que funcionan,* es necesario, además, que sea fácil de mantener para adaptarlo a modificaciones externas. En este sentido, la documentación del proceso es fundamental para hacer un buen desarrollo y es esencial en la fase de mantenimiento del software.

El esquema general, visto anteriormente, se puede detallar dando lugar a modelos concretos del ciclo de vida. A continuación se comentan los modelos de ciclo de vida usados más frecuentemente.

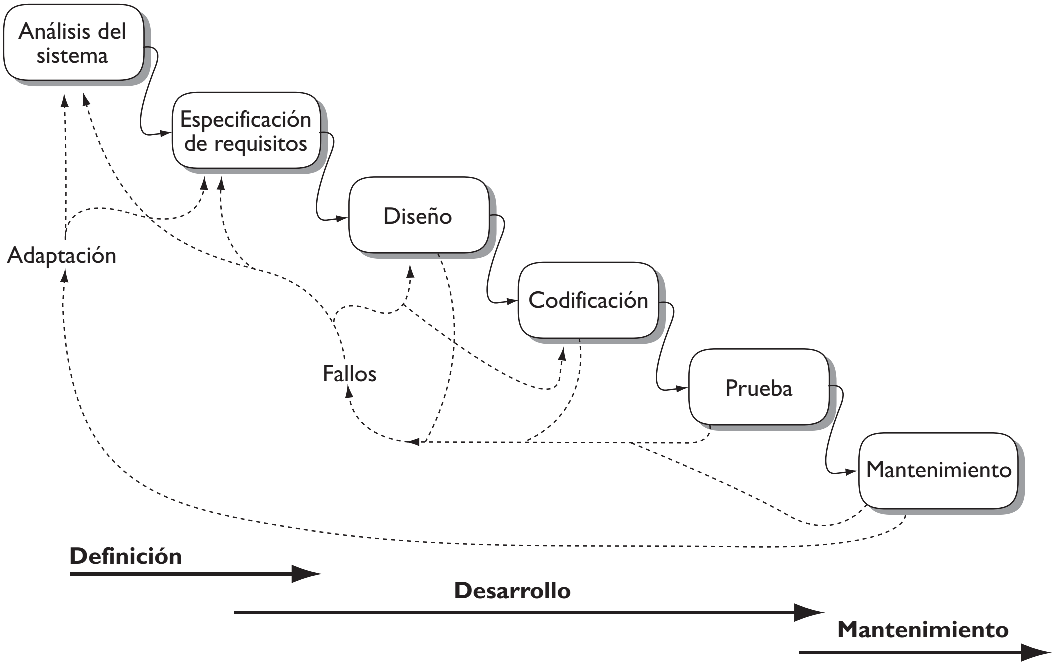


Figura 17.1. *Ciclo de vida clásico.*

17.2.1 Ciclo de vida clásico

El ciclo de vida clásico es un ciclo de fases secuenciales, que sigue un esquema en cascada, análogo al esquema general. Este paradigma contempla las siguientes fases:

- **Análisis del sistema**. El software suele ser parte de un sistema mayor, formado por hardware, software, bases de datos y personas. Por este motivo se debe comenzar estableciendo los requisitos del sistema, asignando funciones a los distintos componentes, y definiendo las interfases entre componentes.
- **Especificación de los requisitos del software.** Antes de comenzar a diseñar el software se deben de especificar las funciones que debe realizar, las interfaces que debe presentar y todos los condicionantes que se puedan presentar, tales como rendimiento, utilización de recursos, etc. Como fruto de este análisis se genera un documento conocido como *especificación de requisitos del software* (SRS).
- **Diseño.** El diseño del software consiste en construir una estructura para el software que permita cumplir los requisitos. Durante esta fase se debe responder a la pregunta ¿cómo se ha de construir el sistema?. El diseño concluye con la confección de un documento de diseño.
- **Codificación.** En la codificación se escriben programas en un lenguaje de programación siguiendo el diseño que se ha realizado previamente. Es decir, se plasma, o traduce, el diseño en programas.
- **Prueba.** Cuando se han escrito los programas, es necesario probarlos. En las pruebas se debe comprobar que los programas se corresponden con el diseño, realizan correctamente sus funciones y que el software satisface los requisitos planteados. Es decir, se ha de contestar a la pregunta ¿se ha construido el sistema que se deseaba?

- **Mantenimiento.** Esta fase se corresponde con el periodo de utilización del software. Dependiendo de la naturaleza y motivación de cada operación de mantenimiento concreta, será necesario revisar desde la codificación, desde el diseño o desde una fase de análisis. La fase de mantenimiento suele ser la más larga y la que más esfuerzo consume. De hecho el mantenimiento se prolonga toda la vida útil del software. Estadísticamente, el mantenimiento suele consumir la mitad del esfuerzo total de desarrollo.

En las fases de definición y construcción, correspondiente al análisis, especificación, diseño, codificación y prueba, el esfuerzo se distribuye de forma desigual. El análisis, la especificación y el diseño requieren aproximadamente un 40% del esfuerzo, la codificación un 20% y la prueba un 40%.

17.2.2 Ciclo de vida de prototipos

Hay situaciones en que no es posible usar el ciclo de vida clásico, fundamentalmente debido a la dificultad de establecer los requisitos del software a priori. En estas situaciones es posible seguir el modelo de **ciclo de vida de prototipos**. En esencia este paradigma se basa en la construcción de un prototipo durante las primeras etapas del ciclo de vida.

Un prototipo es un modelo "a escala reducida" del sistema que se va a construir. El prototipo incorpora sólo los aspectos relevantes del sistema y se utiliza como ayuda en la especificación del software, sirviendo como base tangible sobre la que discutir con el cliente.

El ciclo de vida comienza con la realización de un breve análisis de los requisitos, tras el cual se diseña y codifica el prototipo. Sobre el prototipo se discuten y detallan las especificaciones, modificando el prototipo de acuerdo con éstas, siguiendo un proceso cíclico. El resultado de este proceso es el documento de especificación de requisitos del software.

A partir de este punto se puede desechar el prototipo, diseñándose el sistema definitivo, o pueden realizarse refinamientos sucesivos del prototipo hasta alcanzar un producto utilizable.

17.2.3 Proceso unificado de desarrollo de software

El proceso unificado de desarrollo de software es un modelo de desarrollo asociado al paradigma de **orientación a objetos**. En la última década se han impuesto los lenguajes de programación orientados a objetos, entre los que se encuentran C++ y Java. En estos lenguajes los programas se estructuran en *clases* [Bud94]. Una clase es una abstracción que engloba un estado y un comportamiento. Al nivel del código, un objeto encapsula un conjunto de estructuras de datos y las operaciones ("métodos") que trabajan con ellas, ocultando los detalles de implementación. Además, unas clases se pueden derivar de otras mediante relaciones de herencia, lo que permite obtener subclases por simple especialización de otras ya existentes. El programa trabaja con objetos, pertenecientes a las clases definidas, del mismo modo que en un lenguaje procedural convencional se trabaja con variables pertenecientes a tipos. Para utilizar una clase solamente es necesario conocer los métodos (procedimientos) que puede realizar, del mismo modo en que para utilizar un tipo de datos necesitamos conocer las operaciones que se le pueden aplicar.

Los lenguajes orientados a objetos facilitan el desarrollo del software y aumentan la posibilidad de reutilización del código. Esto es así por la independencia que existe entre el código de las clases y los métodos que se le pueden aplicar, es decir, es posible cambiar la secuencia de instrucciones que implementan un método concreto sin modificar las partes del programa que utilicen ese método. De hecho existen extensas bibliotecas de clases asociadas a estos lenguajes.

Por otra parte, facilitan una transición gradual entre la especificación y el diseño, ya que permiten utilizar los mismos conceptos y notaciones en la especificación, el diseño y la codificación, posibilitando una transición de uno a otro por simple refinamiento de la representación. Por este motivo se han desarrollado modelos de desarrollo basados en técnicas orientadas a objetos, entre estos modelos se encuentra el proceso unificado de desarrollo de software.

El proceso unificado de desarrollo propone un **ciclo de vida cíclico** e iterativo, en el que el software se desarrolla en una secuencia de fases. Cada ciclo acaba con una versión del producto. El primer ciclo corresponde con las fases de definición y construcción del esquema general, mientras que el resto de los ciclos se encuadran en la fase de mantenimiento. Cada ciclo se descompone en fases, que se realizan de forma iterativa. Cada iteración del proceso de desarrollo realiza total o parcialmente una fase. Cada fase acaba cuando se alcanza una meta expresada en términos de los productos que se han generado en ella (documentos, código etc.). Aunque en las primeras iteraciones la mayor parte del trabajo se realiza en especificación y en las últimas se hace en codificación y prueba, en cada iteración se suele realizar trabajo correspondiente a especificación, diseño, codificación y prueba, no obstante, en *el proceso unificado de desarrollo de software* se propone un ciclo de vida descompuesto en cuatro fases: inicio, elaboración, construcción y transición (véase Figura 17.2) [Jac00], y se propone una metodología de desarrollo. El objetivo de la fase de inicio es definir el producto a desarrollar, para lo que es necesario realizar la mayor parte de la especificación, parte del diseño, y tal vez la codificación de algunos componentes críticos. En la fase de elaboración se diseña la arquitectura del software. La fase de construcción tiene por objetivo generar una versión completa del producto. En la fase de transición el software se prueba para generar una versión operacional del mismo.

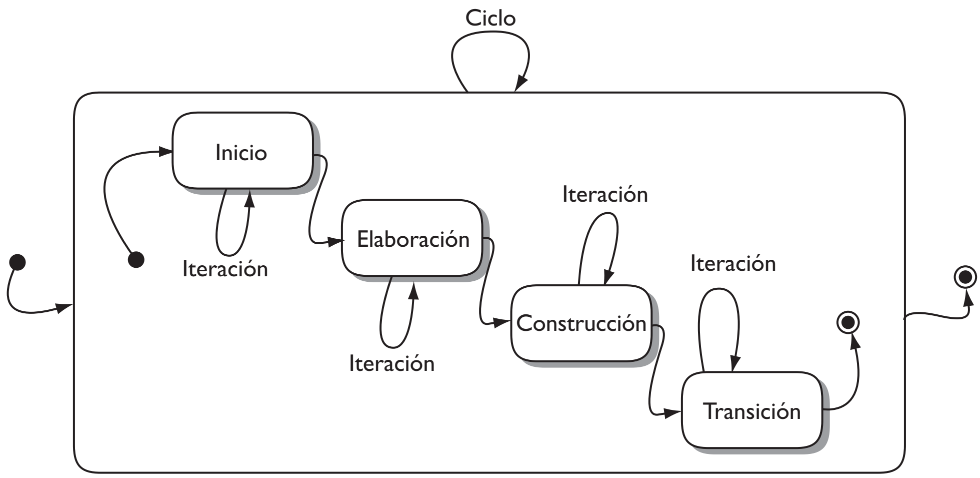


Figura 17.2. *Ciclo de vida en el proceso unificado de desarrollo de software.*

Asociada a la metodología se encuentra el lenguaje unificado de modelado, **UML** (*Unified Modelling Language*), que es un lenguaje gráfico normalizado para expresar de forma visual la especificación y el diseño del software [Boo99]. UML permite describir gráficamente distintos aspectos del software, de un modo equivalente a como un arquitecto describe la estructura de un edificio. La descripción de un sistema en UML incluye una serie de modelos,

referidos a distintos aspectos del sistema, cada uno de los cuales se representa utilizando un conjunto de diagramas específico. Los modelos pueden agruparse en los siguientes:

- **Modelo de comportamiento.** Describe el comportamiento del sistema, detallando las acciones de interacción entre un usuario y el sistema (casos de uso), y la interacción entre objetos del sistema.
- **Modelo estructural.** Describe la estructura de clases y objetos del sistema.
- **Modelo arquitectónico.** Describe los componentes software y hardware del sistema y la distribución de los componentes software sobre el hardware.

17.3 Planificación y gestión de proyectos

El desarrollo de cualquier proyecto real exige la realización de una planificación previa. Esta planificación debe determinar el modelo de ciclo de vida a seguir, los plazos para completar cada fase y los recursos necesarios en cada momento [McC97]. La planificación tiene como objetivos estimar el plazo y coste de desarrollo, y establecer el proceso detallado a seguir, es decir determinar el guión a seguir por todo el equipo de desarrollo durante el proceso. Esta información se recoge en un documento denominado usualmente plan de desarrollo del software. El plan es un elemento fundamental para poder gestionar el proyecto, ya que permite prever los recursos que se van a utilizar, detectar posibles desviaciones en tiempo y presupuesto, y para tomar medidas para corregir estas desviaciones. En muchos casos los proyectos no continúan más allá de la planificación, por ser el coste de desarrollo excesivo o el plazo de realización demasiado largo.

Haciendo una analogía podemos comparar el proceso con el de construcción de una casa. Si no hemos realizado una planificación no sabremos cuánto va a costar, cuánto tardaremos en realizarla, ni cuándo necesitaremos materiales o personal.

Para realizar la planificación es conveniente disponer de una información lo más detallada posible del proyecto a realizar. Es decir, antes de realizar la planificación se debe haber realizado, al menos, un avance del análisis. Dependiendo de las exigencias del proyecto se podrá terminar el análisis antes de hacer la planificación del resto de las fases, o será necesario realizar la planificación contando con una versión preliminar, para después modificar aquélla, si fuera necesario, al completar el análisis. Esta última situación se plantea, por ejemplo, cuando es esencial conocer el coste o la duración del proyecto, con objeto de decidir su viabilidad.

En la realización de un proyecto se usan tres tipos de recursos: humanos, software y hardware. Como **recursos humanos** se incluye a todo el personal que interviene en el proyecto. Normalmente el esfuerzo humano se mide en *personas x unidad de tiempo* (habitualmente en personas por mes), que es la suma del tiempo empleado por todas las personas que intervienen en el proyecto.

Los **recursos hardware** son las computadoras donde se ha de desarrollar el proyecto, las computadoras en la que se usará el software (que pueden no ser las mismas) y los periféricos y dispositivos específicos que se necesitan en ambas máquinas.

Como **recursos software** se incluyen los paquetes y **herramientas software** usados como ayuda durante la realización del proyecto, tanto en la codificación (editores, compiladores, encuadernadores, depuradores simbólicos, bibliotecas, etc.), como en el diseño, especificación y planificación del proyecto.

De entre los tres tipos de recursos, el personal es el más crítico, y el más difícil de estimar. De hecho, la duración y el coste del proyecto dependen esencialmente de los recursos humanos y de la naturaleza del sistema a crear. El esfuerzo necesario para realizar la aplicación aumenta exponencialmente con el tamaño de ésta (véase Figura 17.3). Por otra parte, la relación entre número de personas que intervienen en un proyecto y su duración tampoco es lineal (véase Figura 17.4). Duplicar el personal no implica reducir a la mitad el tiempo. Esto se debe fundamentalmente a los problemas de coordinación y comunicación entre el personal.

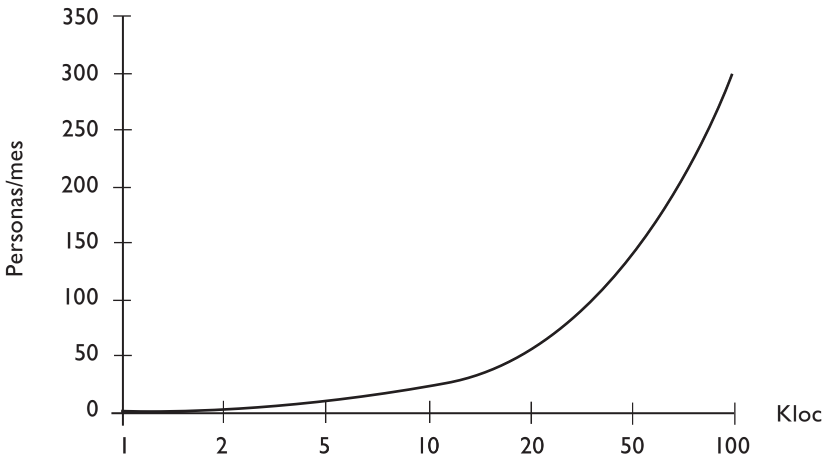


Figura 17.3. *Relación entre el esfuerzo (en personas/mes) y el tamaño de la aplicación (en miles de líneas de código).*

EJEMPLO 17.1

En el desarrollo de una aplicación han intervenido seis analistas durante los cuatro meses que han durado las fases de especificación y diseño, cuatro programadores durante dos meses y un analista y dos programadores realizando pruebas durante dieciocho semanas.

El esfuerzo de desarrollo total ha sido:

Fase	*Nº personas*	*Duración*	*Esfuerzo*
Especificación y diseño	*6 personas*	*4 meses*	*24 p·m*
Programación	*4 personas*	*2 meses*	*8 p·m*
Prueba	*3 personas*	*4,5 meses*	*13,5 p·m*
		Esfuerzo total	***45,5 p·m***

El problema que normalmente se plantea es estimar el número de personas que deben intervenir en el proyecto para acabarlo en una determinada fecha. Para poder resolverlo se necesita cuantificar el esfuerzo necesario para completar el proyecto. Un arquitecto, para determinar el esfuerzo necesario para construir una casa utiliza, junto con datos cualitativos del mismo (por ejemplo, tipo suelo), datos cuantitativos (superficie construida).

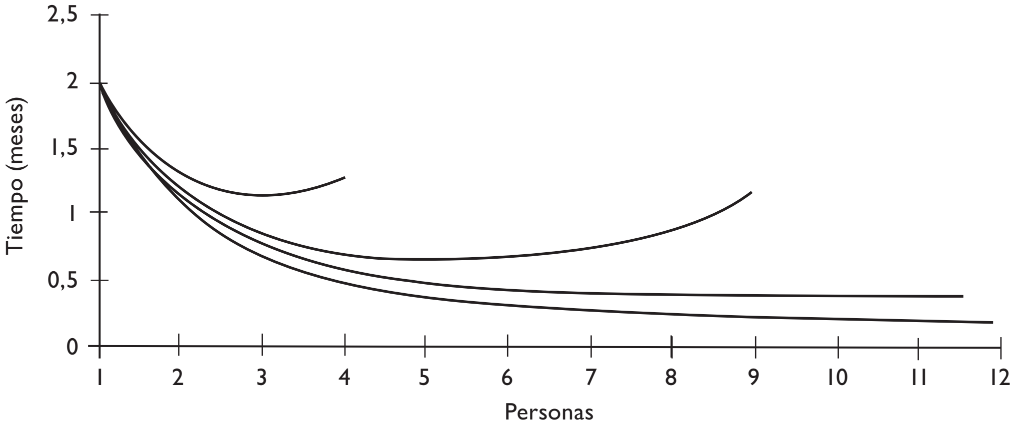


Figura 17.4. *Relación entre el número de personas que intervienen en una tarea y el tiempo empleado en realizarla, para distintos tipos de tareas.*

El establecimiento de medidas cuantitativas del software, que puedan ser estimadas en las fases iniciales del proyecto, es complejo. No obstante, en las últimas décadas se han propuesto diversas métricas para el software. De entre éstas, cabe destacar la utilización como medida del **número de líneas de código** generadas (abreviado LOC, del Ingles *Lines of code*), que si bien no es simple de estimar en las primeras etapas, es fácil de medir a posteriori. La dificultad de realizar la estimación radica, entre otros factores, en que la medida en sí está sujeta a gran número de condicionantes: lenguaje de programación usado, experiencia del personal, organización de éste, etc. No obstante, debe tenerse en cuenta que el objetivo no es *predecir* el esfuerzo de desarrollo o el tamaño del software, si no, tan solo tener una aproximación de su valor que pueda servir de guía al realizar la planificación y el presupuesto. El mismo problema se plantea en otros muchos ámbitos. Cuando un pintor hace un presupuesto para pintar una vivienda, debe estimar el esfuerzo que deberá emplear y la pintura que necesitará. Ambos dependen de la superficie a pintar, por lo que el pintor realizará una *estimación* de la superficie de las paredes y techos de la vivienda, que si tiene experiencia podrá hacer "a ojo". En ningún caso realizará una medida exacta de la superficie, ya que aún así no tendría más precisión en sus cálculos (el consumo de pintura depende de una gran cantidad de parámetros adicionales, como la rugosidad de la pared), y por tanto, no estaría justificado el esfuerzo adicional que se emplearía en obtener una estimación más precisa.

En el caso del software la situación es más complicada, pues código no está ahí para medirlo. La única forma de realizar estimaciones del tamaño del software es adquirir experiencia, y para ello, la mejor estrategia es recoger datos históricos de todas aplicaciones y programas desarrollados (tanto personalmente como en la empresa) [Hum01]. Los datos recolectados incluirán, al menos, el tamaño de la aplicación, el esfuerzo de desarrollo empleado, el personal involucrado, las herramientas usadas, la duración, y una breve descripción del proyecto. Estos datos se podrán usar en futuras estimaciones, y además permitirán analizar la evolución del grado de bondad de las estimaciones previamente realizadas, así como de la productividad.

Se han propuesto varios modelos empíricos que relacionan entre sí diferentes medidas del software, y parámetros del proceso de desarrollo. Usando uno de estos modelos se puede, por ejemplo, estimar el tiempo de realización y el esfuerzo partiendo de una estimación del tamaño de la aplicación, o el tamaño a partir de una medida de complejidad funcional del software.

EJEMPLO 17.2

Se desea desarrollar un simulador de un ordenador didáctico, de acuerdo con la siguiente descripción:

El programa debe realizar una simulación del funcionamiento de un ordenador elemental, permitiendo cargar y salvar programas de éste y hacer volcados de su memoria.

El programa mostrará el panel frontal de ordenador, incorporando sus controles, e incluirá una ventana de visualización del contenido de la memoria, en la que se mostrará el código del programa que se está ejecutando en lenguaje ensamblador. Sobre estas dos ventanas se realiza un scroll automáticamente de forma que la instrucción apuntada por el contador de programa esté en el rango de código visible, y la dirección contenida en el registro de direccionamiento de memoria sea visible en la ventana de memoria. En ambos casos la instrucción/dirección actual se resaltará con un color distinto. El programa incluirá un menú de órdenes, tipo persiana, al que se accederá pulsando la inicial de la orden, y en el que será posible desplazarse usando las teclas de cursor. Además se incluirán órdenes para actuar sobre los conmutadores y pulsadores, activadas por la inicial del nombre de éste, y se podrá visualizar un texto de ayuda. El programa se desarrollará en Pascal, sin usar ninguna biblioteca de gestión de menús.

Para realizar una estimación del esfuerzo de desarrollo se puede hacer una estimación del tamaño del software en líneas de código. Para poder realizar esta estimación haremos un estudio preliminar del sistema, del que se obtendrá una descomposición provisional del software. Supongamos que realizamos la siguiente partición:

Máquina virtual. *Mantiene los valores de los registros, memoria, e interruptores, e interpreta las dieciséis instrucciones de la máquina, actualizando la visualización tras cada microinstrucción.*

Menú. *Subsistema de lectura de menús. Debe realizar la lectura de teclado, interpretando y visualizando los menús desplegables.*

Dibujo. *Módulo de dibujo del ordenador, cuyo estado visualiza a partir de los valores de los registros. Pinta tanto la carcasa como los valores de los LED, luces e interruptores. El dibujo del frontal está compuesto por 50 elementos fijos y 7 móviles.*

Visualización *del contenido de memoria y del programa en ejecución.*

Diálogo. *Módulo de entrada de una línea de texto. Debe permitir la corrección con la tecla de borrado y el movimiento de los cursores.*

Ensamblador. *Interpreta y carga/descarga un texto en ensamblador para el ordenador.*

Registros. *Módulo de actualización de valores de registros.*

Volcado. *Graba en disco el contenido de la memoria.*

Ayuda. *Presentación de ayuda.*

Con esta partición podemos realizar la siguiente estimación de tamaños:

Módulo	***Tamaño (LOC)***	
Máquina virtual	*300*	*15 a 20 LOC * 16 instrucciones*
Menú	*250*	
Dibujo	*600*	*50 elementos fijos a 8 LOC* *7 elementos móviles a 30 LOC*
Visualización	*75*	
Diálogo	*75*	
Ensamblador	*200*	
Registros	*50*	
Volcado	*50*	
Ayuda	*50*	
TOTAL	***1650 LOC***	

Con esta estimación del tamaño, y haciendo uso del modelo COCOMO, podemos estimar el esfuerzo de desarrollo:

$$E = a \cdot T^{b} = 2{,}4 \cdot (1{,}65)^{1{,}05} = 4{,}1\ Personas \cdot mes$$

El proyecto podría ser desarrollado por una persona en cuatro meses.

Uno de los modelos empíricos más simples es el COCOMO (del inglés *COnstructive COst MOdel*), propuesto por Boehm en 1981, que relaciona, entre otros parámetros, el tamaño del software (T, en *KLOC,* miles de líneas de código), con el esfuerzo de desarrollo (E, en personas · mes), mediante la siguiente expresión:

$$E = a \cdot T^b$$

Los parámetros a y b dependen del tipo de sistema a desarrollar. Para aplicaciones simples a vale 2,4 y b 1,05.

La planificación temporal del proyecto se puede representar utilizando un **diagrama de Gantt** (Figura 17.5). En estos diagramas se puede ver qué tareas deben realizarse en un momento dado y las fechas previstas de comienzo y fin de cada tarea.

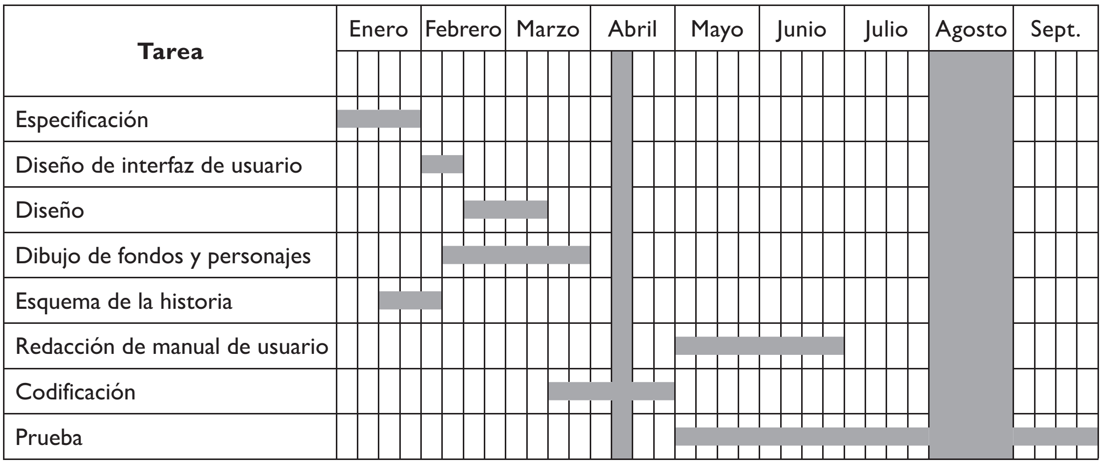


Figura 17.5. *Diagrama de Gantt.*

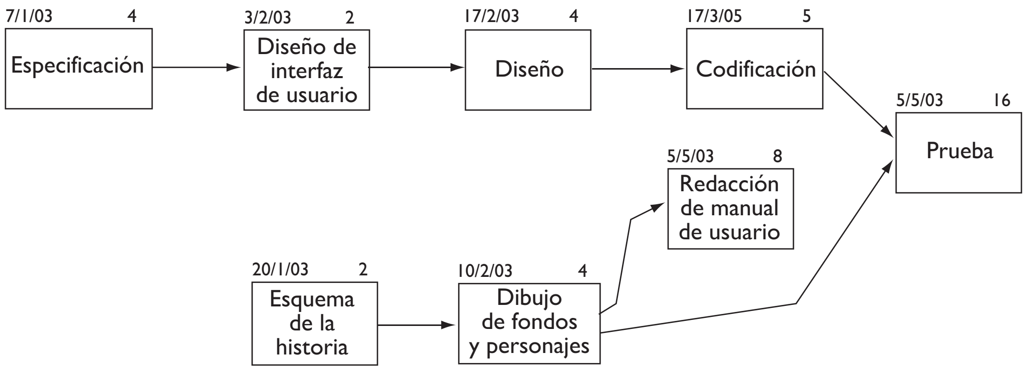


Figura 17.6. *Red de tareas correspondiente al diagrama de Gantt de la Figura 17.5.*

Es frecuente que algunas tareas se realicen en paralelo. En estos casos interesa determinar las dependencias entre tareas y la duración de las mismas, pues el tiempo de realización del proyecto depende sólo de la secuencia de tareas más larga. Para calcular este tiempo se puede dibujar la red de tareas del proyecto (Figura 17.6). Una red de tareas es un grafo

dirigido acíclico que indica las dependencias entre tareas. En el grafo cada tarea es un nodo, dibujándose un arco entre dos tareas cuando una de ellas debe completarse antes de empezar la otra. Sobre este grafo es fácil determinar el camino más largo (camino crítico), del que dependerá el tiempo de realización del proyecto. En la Figura 17.6 se muestra la red de tareas del diagrama de Gantt mostrado en la Figura 17.5, el camino crítico se ha indicado con una doble flecha.

Durante la realización del proyecto, la planificación se contrastará con los tiempos reales de finalización de las distintas tareas, lo que permitirá detectar las posibles demoras en la finalización de cada tarea y predecir su influencia en la fecha de finalización del proyecto.

17.4 Especificación de requisitos del software

Antes de poder diseñar y codificar el software es necesario especificar los requisitos que éste debe satisfacer. Como requisitos del software se deben especificar todas aquellas características del mismo que sean relevantes, incluyendo:

- La función (o funciones) que debe realizar.
- Su interfaz con otros elementos del sistema (personas, hardware y software).
- El modo de desempeñar las funciones (eficiencia, fiabilidad, etc.). lo que habitualmente se conoce como requerimientos no funcionales.

Para determinar estos requisitos, el analista debe realizar una recopilación, estructuración y síntesis de información sobre el sistema a construir. Esta información se obtiene fundamentalmente de *entrevistas* con el cliente o usuario.

Durante el proceso de análisis se redacta el documento de especificación de requisitos software, que pasará a formar parte de la configuración del software. En él debe encontrarse la información necesaria para que el usuario reconozca que ese es el sistema que necesita y para que el diseñador pueda construir el sistema que el usuario desea.

EJEMPLO 17.3

Supongamos que deseamos construir software de control de acceso para una aplicación que funciona sobre un terminal que incorpora un lector de tarjetas (como un cajero electrónico, o un terminal de alquiler de cintas de vídeo), de acuerdo con la siguiente especificación:

El sistema debe leer la tarjeta de cliente, rechazándola si se generan errores de lectura. A continuación comprobará si se ha denunciado el robo de la tarjeta, reteniéndola en caso afirmativo. En caso contrario pedirá un código de identificación al cliente, permitiéndole hasta tres intentos. Si después de tres intentos el cliente no introduce el código correcto el sistema retiene la tarjeta. Si después de un minuto el cliente no ha introducido el código el sistema devolverá la tarjeta. Si la identificación es correcta dará acceso a la aplicación.

Para expresar este funcionamiento podemos usar solamente lenguaje natural, tal como acabamos de hacer, pero éste es más ambiguo, y más difícil de entender que una notación más compacta y más formal. Los requisitos de este ejemplo se expresan con facilidad y concisión con una tabla de decisión. La Figura 17.7 muestra una tabla de decisión que recoge esta especificación. Cada columna corresponde a una combinación de condiciones de entrada. Los guiones en las entradas representan condiciones que no afectan a la acción a tomar. Así, la primera regla se puede leer como: "independientemente del resto de las condiciones, si la tarjeta no se puede leer se debe devolver".

La especificación de requisitos software es finalmente sometida a una revisión con el cliente, tras la cual, si se acepta, comenzaría la fase de diseño.

Una **metodología de especificación** *es un procedimiento para usar sistemáticamente un conjunto de técnicas y herramientas que facilitan la realización de la especificación*. Cada metodología utiliza distintas notaciones para expresar los requisitos, cada una captura determinados aspectos de los requisitos funcionales. Entre las notaciones existentes cabe destacar las tablas de decisión, los diagramas de estados y las diagramas de flujo.

Una *tabla de decisión* muestra las acciones que debe realizar el sistema ante determinadas condiciones de entrada. La primera columna contiene una lista de entradas seguida de una lista de acciones. Las siguientes columnas son reglas que indican combinaciones de entradas y la acción o acciones que deben realizarse cuando éstas se producen (véase Figura 17.7). Las tablas de decisión permiten expresar de forma compacta una parte del comportamiento de un sistema.

		Reglas					
		1	2	3	4	5	6
Condiciones	Tarjeta correcta	N	S	S	S	S	S
	Tarjeta robada	-	S	N	N	N	N
	Más de un minuto en dar clave	-	-	S	N	N	N
	Más de tres intentos de clave	-	-	-	S	N	N
	Clave correcta	-	-	-	-	N	S
Acciones	Volver a pedir clave					X	
	Devolver tarjeta	X		X			
	Retener tarjeta		X		X		
	Mostrar menú						X

Figura 17.7. *Tabla de decisión del Ejemplo 17.3.*

Un *diagrama de estados* es un grafo dirigido en el que cada nodo representa un estado del sistema, y en el que los arcos representan transiciones entre estados, que pueden ocurrir ante condiciones externas, internas, o espontáneamente. Los diagramas de estados son esquemas abstractos del funcionamiento del sistema, por lo que son útiles para modelar su comportamiento. La Figura 17.8 muestra un diagrama de estados para el sistema del Ejemplo 17.3. El estado inicial está indicado por un círculo relleno y el final por un doble círculo.

Un *diagrama de flujo de datos* (DFD) representa la información que se transmite en el sistema de la entrada a la salida (mediante los arcos etiquetados) y las *transformaciones* que se realizan con ella (círculos). (La Figura 17.9 muestra un DFD).

En la especificación de un sistema es habitual utilizar más de una notación. En cualquier caso, la descripción podrá ir refinándose sucesivamente hasta conseguir el nivel de detalle necesario. Así, por ejemplo, un Diagrama de Flujo de Datos se puede ir refinando durante el análisis, a medida que se adquiere un conocimiento más detallado del sistema. Para ello se comenzará realizando un diagrama general o solución a nivel cero. La solución anterior se podrá refinar sustituyendo una transformación por un DFD detallado de la misma. Este proceso se podrá aplicar repetidas veces, obteniendo soluciones sucesivas.

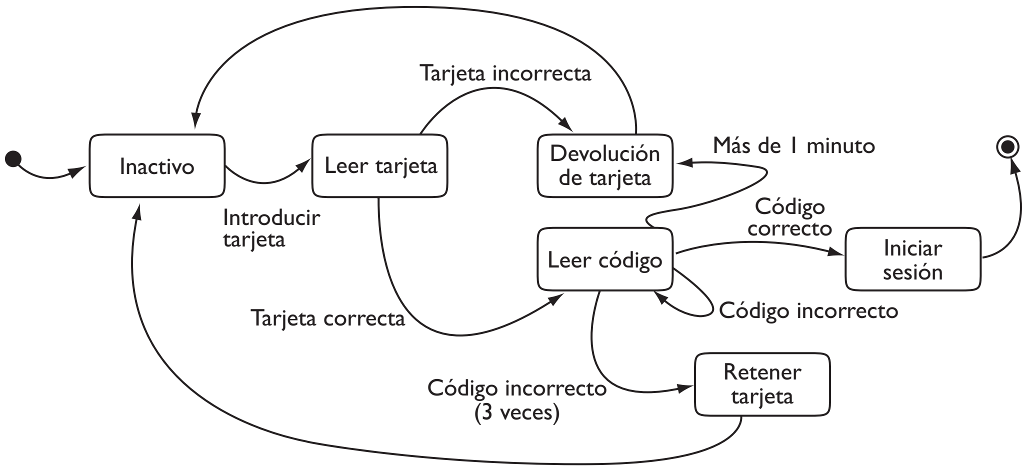


Figura 17.8. *Diagrama de estados del Ejemplo 17.3.*

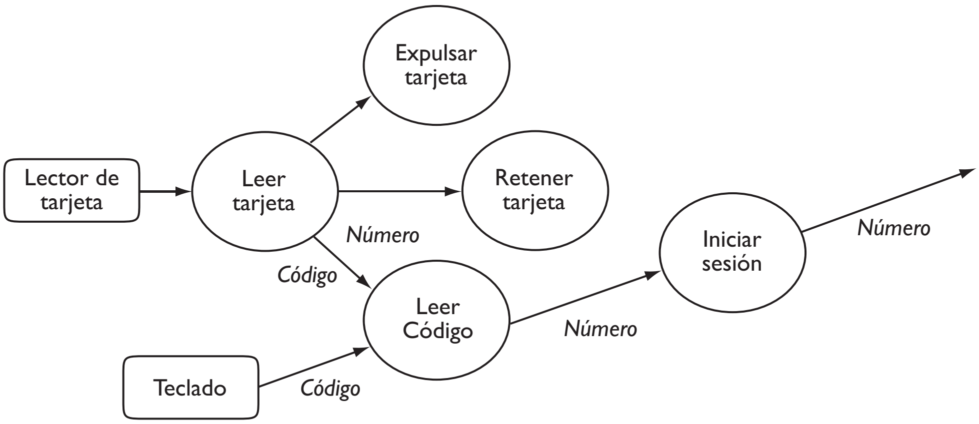


Figura 17.9. *Diagrama de flujo de datos correspondiente al Ejemplo 17.3.*

17.5 Diseño

En él se establece una estructura para el software que permita satisfacer los requisitos. En cualquier ingeniería el diseño es una fase esencial. Antes de comenzar la construcción de un puente, se diseña éste. La calidad del puente depender del diseño que se haya realizado. En él el ingeniero habrá representado la estructura del puente mediante una serie de planos.

El diseño determina "cómo" se va a construir el sistema. El resultado del trabajo de diseño es la *documentación del diseño* que forma parte de la configuración del software.

Una vez concluido el diseño debe ser revisado. Un diseño es bueno si permite la realización ágil de programas que cubran los requisitos, resultando un software fácilmente mantenible.

En el diseño del software se establece la estructura de los programa y de los datos usados por éstos, ambas están íntimamente ligadas. El proceso de diseño debe realizarse comenzando con los aspectos más generales para acabar en detalles concretos; es decir, de forma

descendente. En este proceso se suelen distinguir dos etapas: **diseño arquitectónico** y **diseño detallado**. En la primera se define la estructura general de programas y de datos. Como resultado de esta etapa se obtiene una descomposición del sistema en módulos, de los cuales se especifica claramente su interfaz con el resto del sistema. El diseño detallado se encarga de diseñar los aspectos internos, de procesamiento, de cada módulo y de concretar detalles de bajo nivel de las estructuras de datos.

El resto de este apartado se dedica a estudiar con más detalle aspectos generales del diseño, centrándose en el diseño arquitectónico. Para realizar el diseño detallado de cada módulo se podrán seguir los pasos descritos en la Sección 17.2.

17.5.1 Fundamentos del diseño

Un concepto esencial en el desarrollo del software es el de *modularidad*. La división en módulos facilita la construcción, depuración y mantenimiento del sistema. Una de las tareas a realizar en el diseño arquitectónico es encontrar una división del sistema en módulos que sea satisfactoria.

La división en módulos debe cumplir unos requisitos para ser útil. Los módulos definidos deben ser suficientemente independientes como para permitir un desarrollo en paralelo (por varias personas) y para impedir que la modificación de un módulo afecte a los demás. Si esto se consigue se puede trabajar (en el desarrollo o mantenimiento) con un sólo módulo, reduciéndose la complejidad.

Al igual que en el diseño de algoritmos, un principio básico del diseño del software es la *abstracción*. Durante el proceso de diseño se van definiendo soluciones al problema a distintos niveles de abstracción. A medida que se avanza en el proceso se va reduciendo el nivel de abstracción. En este sentido se debe evitar mezclar distintos niveles de abstracción; esto es, detallar excesivamente unas partes mientras otras permanecen a un alto nivel de abstracción.

Otro principio importante es el de *ocultación de información*. Éste conlleva el que de cada módulo sólo sea "visible" y accesible desde el exterior la información necesaria para comunicarse con otros módulos. El resto de la información del módulo (código y datos) debe ser inaccesible desde el exterior. La ocultación de información dificulta la propagación de errores entre módulos.

17.5.1.1 Estructura del software

Para representar la estructura del software se puede utilizar una gran variedad de herramientas, la más utilizada son las cartas de estructura. Una *carta de estructura* es un diagrama jerárquico en el que aparecen los módulos que forman el sistema (representados como cajas) y las relaciones de utilización existentes entre éstos. Un módulo que utiliza a otro está unido a éste por una línea o flecha, que va del módulo llamante (superordinado) al llamado o subordinado (véase Figura 17.10). La carta de estructura refleja las dependencias entre componentes del software, estableciendo un mapa de él.

La carta de estructura permite la representación del flujo de información entre módulos. Para ello se utilizan pequeñas flechas, que aparecen junto a las líneas de conexión entre módulos. Estas flechas pueden tener en su origen un pequeño círculo. Este círculo origen permite distinguir el tipo de información pasada, entre datos (círculo vacío) y control (círculo relleno). La información pasada se considera que es de control cuando el módulo no la procesa, sino que la utiliza para tomar decisiones.

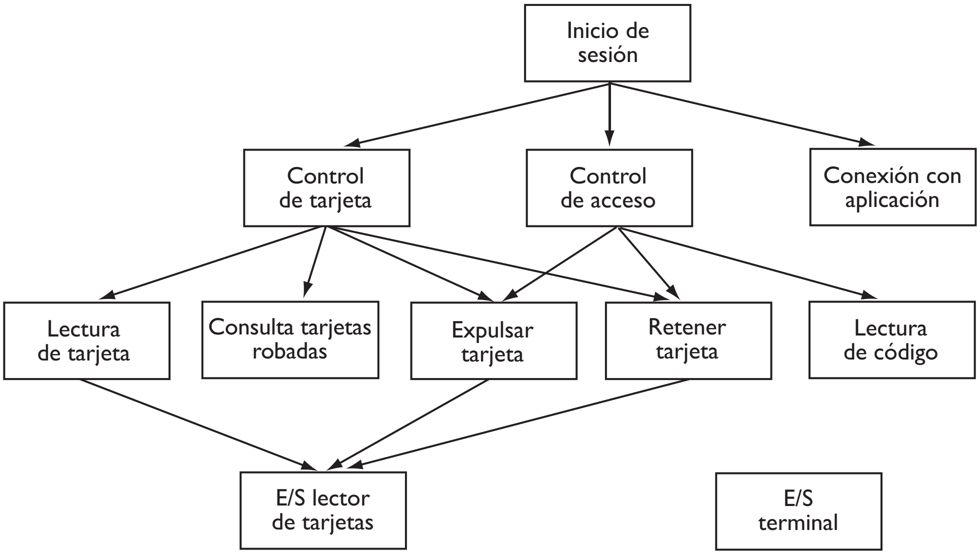


Figura 17.10. *Carta de estructura.*

17.5.2 Relación entre módulos

La independencia entre módulo implica que cada módulo realice una función concreta y presente una interfaz sencilla.

Para determinar de forma objetiva las relaciones entre módulo se definen dos criterios cualitativos: cohesión y acoplo. El **acoplamiento** *es una medida de la dependencia entre dos módulos*. La **cohesión** *es una medida de la concreción funcional del módulo.*

17.5.2.1 Acoplamiento

El nivel de acoplamiento entre dos módulos depende de la interrelación entre ellos. En el diseño se debe pretender que el acoplo entre módulos sea mínimo.

Al aumentar el nivel de acoplo, aumenta la dependencia entre los módulos, y por tanto, la dificultad de prueba y mantenimiento. Un error en un módulo puede propagarse y afectar al otro y una modificación en un módulo puede obligar a modificar el otro.

Lógicamente, no se puede pretender que no exista ningún acoplamiento entre los módulos de un sistema. Todos los módulos que estén relacionados en la carta de estructura tendrán algún tipo de acoplo. No obstante, se debe valorar el acoplamiento entre cada módulo y el resto del sistema y reducirlo al mínimo posible en cada caso. Una especial atención se debe prestar al acoplamiento generado por utilización de datos comunes. Muchos lenguajes de programación distinguen entre datos *locales*, que son accesibles sólo en el procedimiento donde se han definido, y datos *globales* (o comunes), que se pueden usar desde procedimientos distintos al que contiene su definición. En estos lenguajes se pueden utilizar los datos globales para pasar información a un procedimiento, como alternativa a los argumentos. Si se necesita recordar algo entre dos llamadas a un procedimiento, es posible guardarlo en una variable global. El hacer esto, no obstante, implica un alto grado de acoplamiento, y un riesgo de errores grande. La utilización de variables globales, como alternativa a los argumentos, es siempre desaconsejable.

17.5.2.2 Cohesión

La cohesión es una medida de la interdependencia de las distintas partes de un mismo módulo. El nivel de cohesión de cada módulo debe ser lo más alto posible. Un módulo con baja cohesión realizará varias funciones poco relacionadas y deberá descomponerse en varios módulos, de tal forma que cada uno realice una única función.

17.5.3 Metodologías de diseño

Una **metodología de diseño** *es un procedimiento para usar un conjunto de técnicas y herramientas que permitan obtener la estructura del software a partir de la especificación.* Las metodologías de diseño existentes se pueden clasificar en tres grupos, según el enfoque que hacen del sistema:

1. Diseño orientado a flujo de datos.
2. Diseño orientado a las estructuras de datos.
3. Diseño orientado a objetos.

En el primer grupo se utiliza el DFD como punto de partida para la construcción de la carta de estructura. El segundo grupo de metodologías toma como punto de partida el estudio de los datos de entrada y salida del sistema. Entre las metodologías de este tipo cabe destacar la de Jackson y la de Warnier. Este tipo de metodologías son especialmente útiles en sistemas de gestión.

El tercer grupo es el más utilizado actualmente. Se basa en la descomposición del sistema en una serie de clases. Para determinar las clases que es necesario crear en el sistema se suele partir de la definición de casos de uso. En este grupo se encuentra la metodología propuesta en el proceso unificado de desarrollo de software.

17.6 Prueba

Independientemente del proceso de desarrollo seguido, es posible que aparezcan fallos en el software. Los fallos pueden aparecer en cualquier componente, no sólo en el código. Por ejemplo un error en la documentación puede generar errores en el uso, o en el mantenimiento posterior. Con cierta frecuencia el fallo está producido por un error al escribir el programa fuente, pero también puede estarlo por incorrecciones o imprecisiones en la especificación o el diseño.

El objetivo no suele ser, por inviable, demostrar que el sistema software es correcto, si no, asumiendo que siempre se producen fallos, tratar de descubrirlos y eliminarlos. Para descubrir los fallos se sigue un proceso sistemático de búsqueda, centrarse en los elementos, y situaciones en los que sea más probable que el sistema falle.

Los fallos pueden consistir en errores en la lógica del programa, en errores de precisión, en defectos de eficiencia o capacidad, en la documentación, o incluso en defectos en los procedimientos seguidos durante el desarrollo, que dificultarán el mantenimiento del sistema.

Un método usual de detección de fallos es la *inspección*, consistente en la revisión personal del componente. Las inspecciones suele realizarse en grupos pequeños, que estudian y analizan un conjunto de aspectos prefijados de un componente. La inspección se puede realizar con cualquier componente del software. Es importante realizar inspecciones en las fases iniciales de desarrollo, ya que el esfuerzo necesario para subsanar el error aumenta al

avanzar el proceso. Así, por ejemplo detectar un error en el documento de especificación puede suponer tan solo modificar parte de éste, mientras que si el error no es detectado hasta que el sistema ha sido programado será necesario modificar, además, el diseño y rehacer el código.

Una vez que se ha creado el código lo usual es realizar ensayos con él, en una forma semejante a como se realizan ensayos con cualquier otro elemento que se haya diseñado, como por ejemplo, una lavadora. Los ensayos tratan de determinar no solo si la funcionalidad del sistema es la correcta, si no que también deben evaluar los requisitos no funcionales. Para realizar un ensayo de un módulo se deben fijar los objetivos del ensayo, esto es, qué tipos de fallos queremos buscar. En función del objetivo, se diseñarán los casos de prueba. Cada caso de prueba debe indicar los datos a utilizar en el ensayo, y los resultados que se esperan obtener. Existen métodos sistemáticos para derivar los casos de prueba de la especificación.

Una vez realizados los ensayos se deben analizar los resultados, para detectar posibles fallos en el software. Posteriormente se analizarán los fallos detectados para localizar su causa. La complejidad de este proceso aumenta exponencialmente con el tamaño del software que se está probando. Por este motivo, las pruebas se realizan con cada componente aisladamente, para después ir realizando pruebas con bloques de componentes ensamblados (*pruebas de integración*).

Al realizar las pruebas con componentes aislados (por ejemplo, un módulo, una clase, o un procedimiento), es necesario crear un programa ejecutable que incluya el código del componente. Para ello será necesario crear todos sus componentes subordinados ("stub") y un programa principal que lo llame ("driver"). Para no aumentar la complejidad del código a ensayar los módulos subordinados realizados para las pruebas tienen la funcionalidad mínima. El programa principal normalmente acepta datos de entrada y se limita a llamar al procedimiento a probar. La Figura 17.11 muestra el esquema del proceso de ensayo. El código de los drivers y stubs se desecha después de las pruebas. Una vez corregidos los fallos se deben volver a realizar los ensayos, ya que es posible que al corregir alguno se hayan introducidos fallos nuevos.

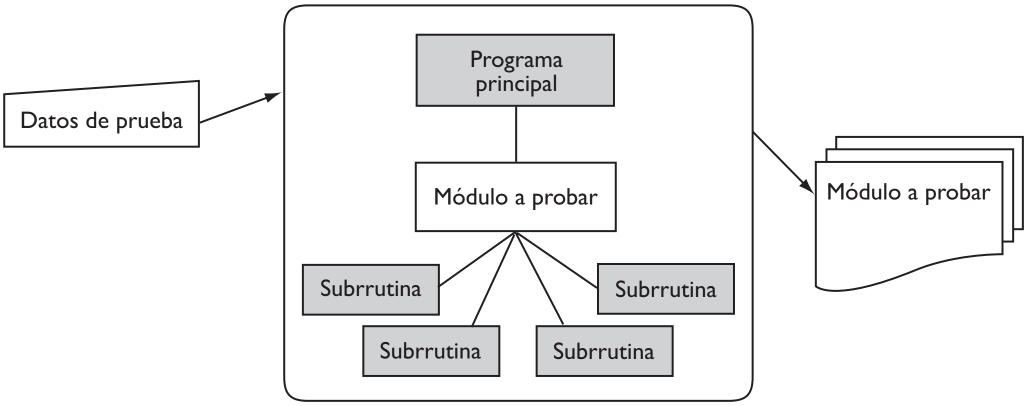


Figura 17.11. *Ensayo de un módulo.*

Una vez probados y corregidos cada uno de los módulos individualmente, se debe ensamblar y probar el sistema. Aunque cada módulo se haya probado individualmente, es posible

que aparezcan fallos al integrarlos, debidos, entre otros motivos a errores en las interfaces entre módulos. Por este motivo, las *pruebas de integración* se suelen realizar de forma controlada y sistemática, facilitando localizar el origen de los fallos que puedan aparecer. Una estrategia usual es integrar los módulos de abajo a arriba ("bottom-up"), según la jerarquía de subordinación. Esta estrategia de integración permite realizar simultáneamente las pruebas de unidad, a nivel de módulo, y las de integración, y además evita la creación de módulos subordinados vacíos para la prueba, ya que en cada momento tendremos ya probados los módulos inferiores de la jerarquía.

La Figura 17.12 muestra el proceso de prueba de un sistema simple cuya carta de estructura tiene tres niveles. En primer lugar se prueban los módulos inferiores (D, E y F), para los que se deberán construir drivers. Una vez probados éstos se prueban los módulos intermedios (B y C), para los que se construirán drivers, y se utilizarán los módulos D, E y F, ya probados y corregidos, como módulos subordinados. Finalmente se ensayará el módulo principal usando como módulos el resto del sistema ya corregido.

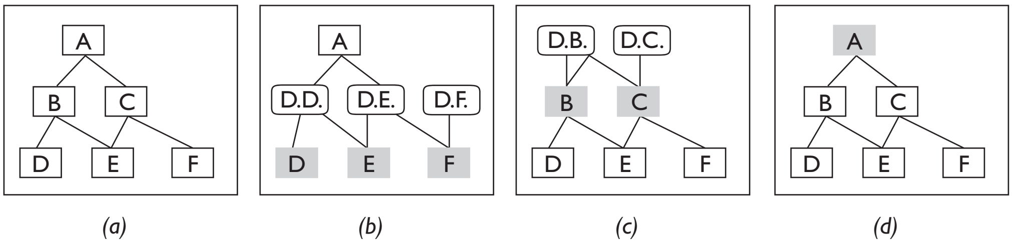


Figura 17.12. *Integración de abajo a arriba ("bottom-up"): (a) carta de estructura; (b) prueba de los módulos inferiores; (c) prueba de los módulos intermedios; (d) prueba del módulo superior.*

Independientemente del proceso de prueba seguido, no es posible garantizar que el sistema está libre de fallos. Por este motivo es importante establecer criterios de terminación de las pruebas que nos den una idea de la probabilidad de que queden fallos sin detectar en el sistema. Una técnica simple para conseguir esto es utilizar dos equipos independientes para realizar las pruebas, estos equipos realizarán ensayos y buscarán errores de forma independiente (la corrección de estos errores será realizada una sola vez posteriormente). Cada uno de los equipos encontrará fallos diferentes de entre los existentes en el software. Esto nos permite calcular la probabilidad de encontrar fallos de cada un de estos equipos, y utilizar esta probabilidad para estimar el número de fallos que quedan en el sistema sin detectar, y la probabilidad de que no queden fallos en el sistema.

17.7 Conclusión

En este capítulo se han abordado conceptos básicos en la construcción del software. Se ha pretendido dar una visión global, del proceso, desde la planificación hasta la prueba. Se ha pretendido, tan sólo, introducir los problemas, el estado actual, las técnicas y sus posibilidades. Para ampliar este tema se remite al lector a alguna monografía sobre ingeniería del software [Pfl98, Pre97, Som92]. Para introducirse en la programación dirigida a objetos puede consultarse el libro de Budd [Bud94].

EJERCICIOS

17.1 Cuando se está usando el modelo de ciclo de vida de prototipos, ¿tiene sentido dejar el prototipo al cliente como producto utilizable?. Razonar sobre las ventajas e inconvenientes de utilizar prototipos.

17.2 Estimar el esfuerzo necesario para realizar el proyecto cuya red de tareas aparece en la Figura 17.6, suponiendo que en la especificación intervienen dos personas, en el diseño tres, en la codificación cuatro, dos en la redacción del guión y del manual de usuario y ocho en el dibujo de fondos y personajes.

17.3 Supongamos que estamos realizando el proyecto cuyo digrama de Gantt aparece en la Figura 17.5, y que a primeros de Marzo detectamos que el diseño va a terminar dos semanas tarde. ¿Qué efecto tendrá en el proyecto? ¿Cómo podemos reducir el impacto de este retraso?.

17.4 Completar la especificación del Ejemplo 17.2.

17.5 Dibujar la carta de estructura para el programa "Indice", que aparece en el Ejemplo 3.10.

17.6 Sobre la carta de estructura realizada en el ejercicio anterior valorar la cohesión y acoplo entre módulos. ¿Qué modificaciones podrían realizarse para mejorar el diseño?

17.7 Realizar el análisis de requisitos y el diseño del supuesto del Ejemplo 3.10.

17.8 Modifica la tabla de decisión de la Figura 17.7 para incluir una opción para que el usuario pueda cancelar la operación en cualquier momento, salvo que la tarjeta sea robada.

17.9 Modificar el diagrama de estados de la Figura 17.8, para permitir que se cancele la operación únicamente después de haber introducido una sola vez un código incorrecto (*Sugerencia:* separar el estado leer código en tres, leer primera vez, leer segunda y leer última).

17.10 Estima y recoge datos de todos los programas que realices, incluyendo el tamaño y el esfuerzo dedicado.

17.11 ¿Qué métrica usarías para estimar el tamaño de una interfaz de usuario?

17.12 ¿Cómo afectan los comentarios a la medida de líneas de código? ¿Y el estilo de programación?

17.13 Haz una estimación del esfuerzo, usando el modelo COCOMO, de un programa que ya hayas hecho y del que sepas el esfuerzo real que has empleado. Comenta el resultado.

BIBLIOGRAFÍA

[Boo99] Booch, G.; Rumbaugh, J.; Jacobson, I.: "El lenguaje unificado de modelado". Addison-Wesley, 1999.

[Bud94] Budd, T.: "Introducción a la programación orientada a objetos", Addison-Wesley, 1994.

[Hum01] Humphrey, W. S.: "Introduction to the personal software process". Addison-Wesley, 2001.

[Jac00] Jacobson, I.; Booch, G.; Rumbaugh, J.: "El proceso unificado de desarrollo de software". Addison-Wesley, 2000.

[McC97] McConnell: "Desarrollo y gestión de proyectos", McGraw-Hill, 1997.

[Pfl05] Pfleeger, S. L.; Atlee, J. M.: "Software Engineering. Theory and practice". Prentice Hall, 2005.

[Pre02] Pressman, R.: "Software Engineering A Practitioner's Approach", 6th Edt. McGraw-Hill, 2005.

[Pri05] Prieto, A.; Prieto, B.: "Conceptos de Informática", Serie Schaum, McGraw-Hill, 2005.

[Som04] Sommerville, I.: "Software Engineering". 7th Edt., Addison-Wesley, 2004.

CAPÍTULO 18

Transmisión y comunicación de datos

Un computador, como sistema de procesamiento, debe intercambiar unidades de información entre sus unidades locales o incluso entre distintos computadores separados a grandes distancia. Esta información, sea cual sea su naturaleza u origen, se conoce con el nombre genérico de **dato**, y, en el contexto de los computadores, los datos están formados por símbolos binarios o bits bajo la forma de ceros y unos. Debido al desarrollo de las redes de computadores y a la informática distribuida, en general, cada vez con más frecuencia las unidades o computadores a interconectar se encuentran a distancias mayores. Esto presenta problemas específicos que se abordan en el presente capítulo, y que se suelen agrupar bajo el término de transmisión y comunicación de datos. Mientras el concepto de **transmisión de datos** hace referencia a la *transferencia* de datos entre dos puntos, la **comunicación de datos** se corresponde con el hecho de *compartir* e *intercambiar* datos. En este capítulo, en primer lugar (Sección 18.1) planteamos los objetivos, las necesidades que se pretenden cubrir y las posibilidades de las redes de comunicación. El resto del capítulo (Sección 18.2) lo dedicamos a los fundamentos de la comunicación de datos. Así, en secciones sucesivas, se trata de la naturaleza de las señales que "transportan" los datos, los tipos y características de los medios o canales de comunicación (Sección 18.2.2): cables de hilos de cobre, cables coaxiales, fibra óptica y radiofrecuencia. Para poder transportar la información es necesario realizar un proceso de **modulación** de una señal portadora con el mensaje (datos) a transmitir, existiendo distintas posibilidades de modulación que se estudian en la Sección 18.2.3. El siguiente apartado, Sección 18.2.4, se dedica a la **multiplexación**, que es una técnica con la que se logra aprovechar con gran eficiencia la capacidad de un canal de comunicación. En general, la transmisión de datos a distancias no muy cortas (mayores de unos pocos metros) se realiza en serie, ya que el realizarla en paralelo supondría la utilización de múltiples canales simultáneamente; por ello la Sección 18.2.5 se dedica a **comunicaciones serie**. El capítulo concluye (Sección 18.2.6) presentando algunas técnicas o algoritmos utilizados para el **control de errores** en comunicaciones digitales.

18.1 Objetivos de las redes de comunicaciones

Una **red de comunicaciones** es un conjunto de equipos y facilidades para el transporte de información entre usuarios ubicados en varios puntos geográficos. En el contexto de éste libro nos referimos a comunicaciones de tipo eléctrico y óptico. El origen de las

telecomunicaciones, o comunicaciones electrónicas a distancia, se remonta a Samuel Morse que inventó el telégrafo en 1844. Con el telégrafo se podían transmitir por medio de señales eléctricas mensajes de forma casi instantánea entre grandes distancias. En 1876 Alexander Bell amplió las posibilidades del telégrafo ampliándolas a la transmisión de señales de voz, surgiendo así el teléfono. Posteriormente se desarrollaron otros medios de comunicación electrónica para difusión de la información, como la radio y la televisión. En la actualidad ha adquirido un gran auge el servicio telefónico inalámbrico (telefonía celular); pero sin duda, el impacto económico y social de mayor relevancia de las telecomunicaciones se centra en las redes de computadores, que permiten desde una pequeño computador (PC o computador móvil) en cualquier hogar intercambiar datos y compartir recursos hardware costosos con otros computadores ubicados a lo ancho de la superficie terrestre.

La información que se puede transmitir a través de una red de telecomunicaciones es de origen muy variado. Por lo general el concepto de **comunicación de datos** se asocia al intercambio de información que previamente ha sido codificada en forma digital (representada por medio de ceros y unos o en varios niveles discretos) entre dos dispositivos a través de un medio (cable, por ejemplo). Entre los servicios proporcionados por las redes de comunicaciones se encuentran los siguientes:

- Telefonía.
- Radiodifusión y televisión.
- Telefonía celular.
- Redes de computadores:
  - Acceso remoto a un computador para el procesamiento de aplicaciones.
  - Transferencia de archivos.
  - Localización de archivos.
  - Correo electrónico.
  - Aplicaciones web (acceso a documentos distribuidos en computadores de todo el mundo).
  - Video bajo demanda.
  - Audio conferencias.
  - Vídeoconferencias.
  - Telefonía a través de Internet.

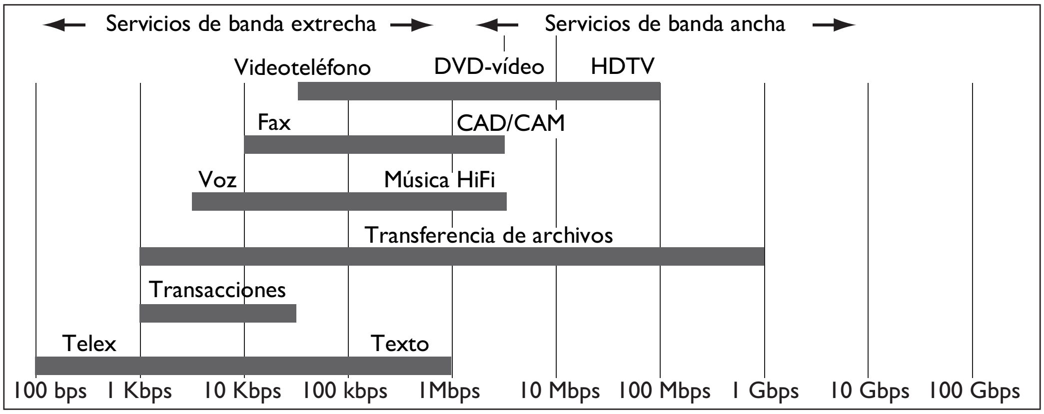


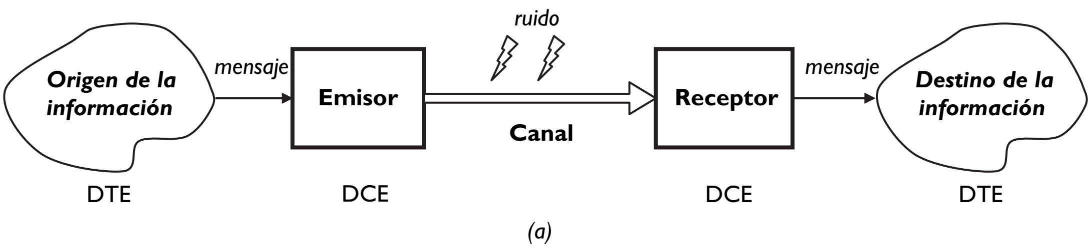
Figura 18.1. *Rangos de tasas de datos requeridas por diversas aplicaciones (extraído de For02).*

Cada una de estas aplicaciones requiere unas determinadas prestaciones en cuanto a corrección de la entrega (al destino establecido), tiempo de duración de la transferencia (velocidad de transferencia o **tasa de datos**, en bits por segundo), precisión de la información, fiabilidad y costo. En la Figura 18.1 se incluyen una lista de servicios agrupadas por requisitos de tasa de datos [For02]. Las aplicaciones que requieren tasas de datos menores de 100 Kbps se dice que son de banda estrecha y los de más de 1 Mbps de banda ancha. Hay que hacer constar que, conforme se van perfeccionado las técnicas de compresión de datos (Sección 4.7) los requisitos en cuanto a tasa de datos pueden ir disminuyendo; así, en la actualidad el DVD-vídeo en resolución de 720x480 requiere unas tasas de datos entre 4 y 9 Mbps, aunque se han desarrollado nuevos formatos de compresión (como WM9) con el que con ese mismo rango de velocidades se puede transmitir video de alta resolución (1920x1080). La TV de alta definición (HDTV) estándar requiere una tasa de datos de 19,4 Mbps.

18.2 Fundamentos básicos de la comunicación de datos

Un **sistema de comunicación** es un sistema que transmite información desde un lugar (**emisor** o **transmisor**) a otro lugar (**receptor**).

En la Figura 18.2a se muestra un esquema simplificado de un sistema de comunicación. A la información que se que pretende llegue al receptor ("información con significado" para el destinatario) se denomina **mensaje**.



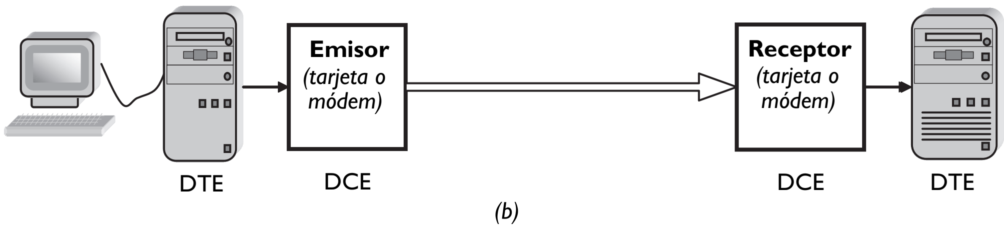


Figura 18.2. *(a) Esquema simplificado de un sistema de comunicación; (b) esquema simplificado de un sistema para transmisión de datos.*

El emisor transforma el mensaje original en señal eléctrica, por medio de un transductor (un micrófono, por ejemplo), y la adecúa y amplifica obteniendo otra señal (**señal a transmitir**) apta para ser eficientemente emitida a través del canal y captada por el receptor. En el receptor se realiza el proceso inverso: la señal transmitida se reconvierte en el mensaje de forma que ésta sea inteligible para el receptor final. Si el canal fuese una fibra óptica el proceso sería similar, ya que se utilizarían señales ópticas en lugar de señales eléctricas.

Hay muchos sistemas de comunicación, tales como el teléfono, que son bidireccionales; en ellos en ambos puntos de conexión existe transmisor y receptor, intercambiándose las funciones de ambos de forma que siempre el extremo que actúa como emisor se conecte al receptor del otro extremo.

Los sistemas de comunicación a que se refiere este capítulo son sistemas en los que tanto el emisor como el receptor son equipos electrónicos. La señal transmitida puede ser de naturaleza eléctrica (tensión o corriente), electromagnética, u óptica.

En los sistemas de comunicación de datos (Figura 18.2b) en el emisor y receptor suelen distinguirse dos equipos:

- **Equipo terminal de datos o DTE**, que son los elementos origen y destino finales de la información.
- **Equipo terminal de la línea de comunicaciones o DCE**, que es el emisor o receptor, propiamente dichos; encargados específicamente de adaptar la señal o mensaje para transmitirlo o recibirlo convenientemente.

Los equipos terminales de datos también se denominan **estaciones** o **sistemas huésped** (*host systems*).

En este apartado, a continuación, se consideran los elementos básicos que intervienen en la transmisión de datos; es decir:

- Las señales a transmitir (Sección 18.2.1), y
- Los medios o canales de transmisión (Sección 18.2.2).

Por otra parte, casi siempre para efectuar una comunicación de datos eficiente es necesario realizar ciertas transformaciones de las señales originales, denominadas modulación y multiplexación. El análisis de estas transformaciones será el objetivo de las Secciones 18.2.3 y 18.2.4.

Debido al coste de las líneas de comunicación, la transmisión de información en redes de comunicación se efectúa en serie; es decir, en un instante dado un emisor transmite (y el receptor recibe) un único bit de información. La Sección 18.2.5 describe algunos conceptos básicos sobre este tipo de transmisión.

Este apartado concluye con la Sección 18.2.6 que trata de la introducción de redundancias en la información binaria, con objeto de poder detectar en el receptor si la información emitida ha sido tan deteriorada al circular por la línea de transmisión que es reconocida erróneamente por el receptor.

18.2.1 Señales a transmitir

A continuación se van a efectuar unas consideraciones teóricas sobre los mensajes, de gran importancia en la caracterización de sistemas de comunicaciones de equipos informáticos.

Los mensajes se caracterizan por un **contenido o cantidad de información**. Un símbolo, cuanto menor probabilidad tiene de producirse, mayor información contiene. Así, si se emite un símbolo (carácter, fonema, etc.), cuya probabilidad de producirse es p, la cantidad de información que contiene se define como:

[18.1]

$$c(p) = -\log_2(p) \text{ bits}$$

Se utiliza como unidad de información el **bit**; esta información corresponde a un símbolo tomado de entre dos equiprobables (símbolos 0 ó 1; $p_0 = p_1 = ½; c(p) = -\log_2(½) = 1$ bit).

No debe confundirse el concepto de bit dado anteriormente como unidad de cantidad de información, que está ligado a la probabilidad de que se transmita un determinado símbolo, con el concepto de bit dado en Informática (un valor binario, 0 ó 1).

EJEMPLO 18.1

Para poner de manifiesto esta diferencia entre el bit definido en el Capítulo 1 y el bit definido en Teoría de la Información, supóngase que se tienen tres símbolos, A, B y C, con probabilidades de producirse 1/2, 1/4 y 1/4, respectivamente. Se pueden codificar según se indica en la tercera columna de la Tabla 18.1 que se da a continuación. El contenido de información de cada símbolo, $c(p_i)$, será el que se indica en la cuarta columna de la tabla.

Tabla 18.1. *Ejemplo que pone de manifiesto la diferencia entre bit y dibitio.*

Símbolo	p_i	Código	$c(p_i)$
A	½	01	1 bit
B	¼	10	2 bits
C	¼	11	2 bits

El símbolo A contiene 1 bit de información, sin embargo está codificado con dos bits.

En el contexto de la Teoría de la Información al bit que especifica un valor binario se le denomina **binit**, cuando se puede dar lugar a confusión con la unidad de cantidad de información. Según esta terminología se dice que el símbolo *A* del ejemplo anterior contiene un bit de información y está codificado con *2 binits*.

EJEMPLO 18.2

Sea una pantalla de un terminal con 768 ·1024 = 786.432 puntos de imagen, con 16 atributos por punto de imagen. Suponiendo que los 16 atributos son equiparables, la cantidad o contenido de información de cada imagen sería:

$$c = -786.432 \cdot \log_2(1/16) = 786.432 \cdot \log_2(16) =$$
$$= 786.432 \cdot \log_{10}(16)/\log_{10}(2) = 946.958{,}49/0{,}30103 =$$
$$= 3.145.728 \text{ bits}$$

Además de por su contenido de información, los mensajes pueden caracterizarse por su ancho de banda. Según el análisis de Fourier [Car86], toda señal *s(t)* que evoluciona en el tiempo (Figura 18.3a) puede descomponerse en un conjunto de infinitas señales sinusoidales de distintas amplitudes máximas *A(f)* y distintas frecuencias *f*; en otras palabras es lo mismo considerar la señal original que la suma de las señales sinusoidales correspondientes. Cada una de las señales sinusoidales que componen la señal original se conoce como **armónica**. Al conjunto de amplitudes y frecuencias *[A(f),f]* de una señal se le denomina **espectro de amplitudes** de esa señal. (Figura 18.3b).

Si la señal es periódica e infinita (Figura 18.3c) el espectro (Figura 18.3d) es discreto: hay infinitos armónicos pero distribuidos discretamente a lo largo del eje *f*. En general una

señal no periódica (Figura 18.3a) tiene un espectro (Figura 13b) continuo (hay armónicos para todas las frecuencias).

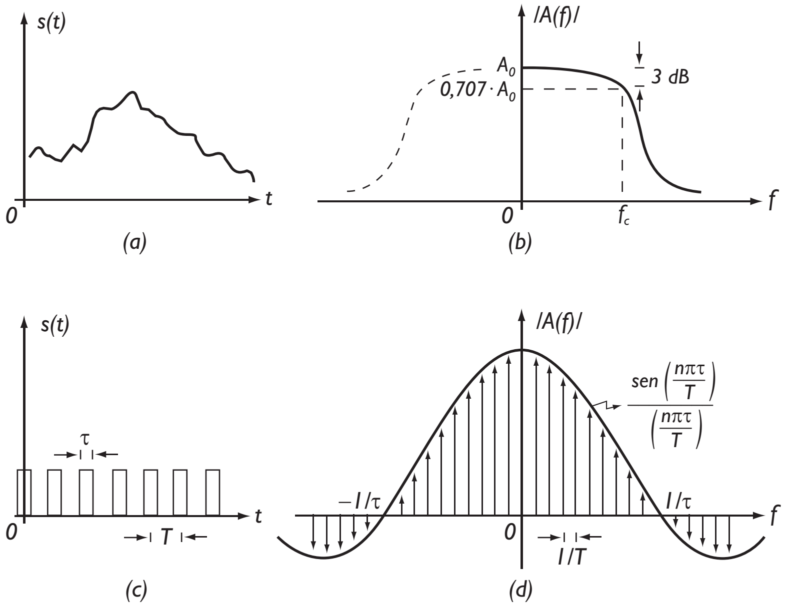


Figura 18.3. *(a) Señal continua variable en el tiempo; (b) espectro de amplitudes; (c) tren de pulsos de grosor τ y período T; (d) espectro de amplitudes del tren de pulsos.*

En la Figura 18.3 puede observarse que hay regiones o bandas de frecuencia para las que las amplitudes de los armónicos son mayores y, por tanto, en la suma de todos ellos contribuirán más que los armónicos alejados de dicha banda, cuyas amplitudes tienden asintóticamente a cero. Ello permite aproximar el espectro total por una zona o banda de él. Se toman como frecuencias límites, f_c, de la banda, aquellas para las que las amplitudes máximas caen a 0,707 A_0; donde A_0 es la amplitud máxima. Es decir:

[18.2]

$$f_c \text{ es tal que } |A(f_c)| = 0{,}707 \cdot A_0, \text{ ó } 10 \cdot \log|A_o/A(f_c)| = 3 \text{ dBs}$$

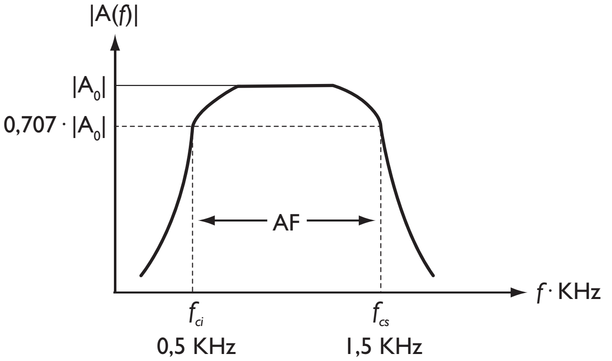


Figura 18.4. *Ancho de banda de una señal.*

Usualmente (Figura 18.4) se consideran dos frecuencias límites (o **frecuencias de corte**), una inferior, f_{ci}, y otra superior, f_{cs}; pudiendo ser f_{ci}=0.

De esta forma se puede aproximar el espectro original, que contiene frecuencias de $-\infty$ a $+\infty$, por un espectro finito, con frecuencias f que cumplen: $f_{ci} \leq f \leq f_{cs}$.

Se denomina **ancho de banda** (**AF**) de una señal al conjunto de frecuencias del espectro de dicha señal comprendido entre las frecuencias de corte. Es decir, es el rango de frecuencias en las que no hay una atenuación significativa de la señal, donde se concentra la mayor parte de su energía. El ancho de banda de la señal de la Figura 18.4 es:

$$AF = 1{,}5 - 0{,}5 = 1 \text{ KHz}$$

y está comprendido entre f_{ci} = 0,5 KHz y f_{cs} = 1,5 KHz.

En la Tabla 18.2 se incluyen los anchos de banda correspondientes a diversas señales analógicas.

Tabla 18.2. *Ancho de banda de algunas señales.*

Señales analógicas	– Voz humana y música	0,1 a 22 KHz
	– Voz a través de teléfono	0,2 a 3,4 KHz
	– Señal de vídeo (TV)	0 a 4,6 MHz

En transmisión de datos digitales en lugar de utilizar la frecuencia para designar el número de cambios por segundo de una señal, se utiliza la **tasa de datos** (*bits rate)*, r, que se da en bits/segundo o bps. En cierta medida la tasa de datos puede considerarse como una **tasa de datos**, ya que nos indica el número de bits que se transmiten por unidad de tiempo.

Hay sistemas de codificación de información en los que cada cambio (o señal elemental o nivel) en señal transmitida representa más de un bit; en este caso el número de cambios de señal por segundo se denomina **tasa de baudios**, R.

EJEMPLO 18.3

Una transmisión de información se realiza a una tasa de bits de 4.000 bps, si en cada cambio de la señal se codifican 4 bits, la tasa de baudios será:

[18.3]

$$R = \frac{r}{q} = \frac{4.000}{4} = 1.000 \text{ baudios}$$

Como veremos en la Sección 18.2.3.4 existe una relación directa entre ancho de banda de una señal analógica y la tasa de datos requerida para ser captada en el receptor adecuadamente. Así en el Ejemplo 18.7 mostraremos que para transmitir una señal telefónica de voz con un ancho de banda de AF = 4 KHz y ser recibida correctamente, debe transferirse a una Tasa de 64Kbps. En otros casos también resulta fácil calcular las tasas de datos de distintos tipos de señales digitales, como se pone de manifiesto en el siguiente ejemplo.

EJEMPLO 18.4

Obtener la tasa de datos requerida para una señal de vídeo con resolución SVGA y con una cadencia de 30 cuadros/segundo.

Según indicamos en la Tabla 8.6, la resolución SVGA es de 800 x 600 puntos de imagen, y, suponemos, que el color de cada punto se codifica con 8 bits. De acuerdo con lo anterior, el número de bits que componen un cuadro (una imagen) es el siguiente:

$$C_c = 800 \cdot 600 \cdot 8 = 3{,}66 \text{ Mbits}$$

Como se han de transmitir 30 imágenes por segundo, la tasa de datos será:

$$r_{TV} = 3{,}66 \cdot 30 = 110 \text{ Mbps}$$

Es conveniente indicar, como vimos en la Sección 4.7, que, si se utilizan técnicas de compresión de datos, esta cantidad podría reducirse notablemente (al orden de 2 Mbps).

En la Figura 18.1 se muestran rangos de tasas de datos requeridas para distintos tipos de aplicaciones.

18.2.2 Medios de transmisión

El canal o medio de transmisión en una red de comunicaciones de datos suele estar constituido por elementos de los siguientes tipos:

- Pares trenzados, para transmisión de señales de tensión o corriente.
- Cable coaxial, formado por un conductor central separado por un aislante de una funda o camisa conductora externa, tal como se utiliza para distribuir las señales de TV (antena, por ejemplo).
- Fibra óptica cilindro de vidrio muy fino (núcleo) recubierto por otro cilindro concéntrico también de cristal (revestimiento), y protegido del exterior por una funda de plástico. Las señales se transmiten en forma de radiación óptica.
- La atmósfera y espacio libre para transmisión de señales electromagnéticas (enlaces de radio, telefonía celular, etc.).
- Luz infrarroja, para distancias muy cortas (1 a 8 metros).

Las señales transmitidas en el medio pueden ser alteradas indeseablemente por dos motivos:

a) *Atenuación.* En efecto, las señales con la distancia se debilitan y deforman, siendo muchas veces necesario incluir en el canal, a ciertos intervalos de distancia, **repetidores** que reconstruyan y amplifiquen la señal original.

b) *Ruido* que interfiere con la señal transmitida, pudiendo incluso imposibilitar al receptor recuperar el mensaje original. Cuanto más potente sea la señal transmitida, menos le afectará el ruido. Si la potencia de la señal emitida es S y la del ruido es N, se denomina **relación señal-ruido** al cociente S/N. Esta relación suele darse en decibelios:

[18.4]

$$10 \cdot \log_{10}\left(\frac{S}{N}\right) \text{ decibelios}$$

Desde un punto de vista técnico dos aspectos o factores de gran importancia en un sistema de comunicación son:

1) *Fiabilidad en la transmisión del mensaje.* Es decir, el sistema debe proporcionar en el elemento destino el mensaje sin errores, tal como se introdujo en el emisor. Hay sistemas que necesitan mayor fiabilidad que otros, tal como puede ser el sistema de transmisión de datos de una entidad bancaria frente al de transmisión de una conversación telefónica.
2) *Tasa de datos.* En general interesa que la tasa de datos de la información por el canal sea la mayor posible. En cualquier caso ésta debe adecuarse a la naturaleza de los entes que se intercomunican. Así la velocidad de transferencia de información entre dos computadores debe ser superior a la que se establezca entre un terminal bancario y una impresora de caracteres.

En la Tabla 18.3 se incluyen algunos ejemplos de las tasas de datos establecidas para canales de transmisión dentro de distintos medios. Los datos se refieren a distintas normalizaciones o redes, la mayoría de las cuales se analizarán a lo largo de este capítulo.

Los sistemas electrónicos y los medios físicos pueden transmitir señales sólo dentro de determinadas bandas de frecuencias, de forma que los armónicos de la señal procesándose o transmitiéndose que estén fuera de esa **banda pasante**, o superen el ancho de la misma (**ancho de banda** *AH*) se atenuarán, distorsionándose la señal o incluso pudiéndola hacer irreconocible al receptor. Obviamente, por tanto, hay que adecuar la señal a transmitir a las características del canal; entre otras, el ancho de banda de la señal a transmitir, AF, no debe ser superior al ancho de banda del canal (y de los equipos emisor y receptor):

[18.5]

$$AH > AF$$

Otro factor que afecta al canal como medio transmisor es el ruido, ya citado anteriormente. Cuanto más potente sea la señal transmitida, menos le afectará el ruido.

Shanon definió la **capacidad de información de un canal** como la máxima tasa de datos teórica. Si *AH* es la banda pasante del canal, la capacidad del canal para una señal que se transmite con *n* niveles viene dada por el Teorema de Nyquist que establece:

[18.6]

$$C = r_{máx} = 2 \cdot AH \cdot \log_2(n)$$

Conviene hacer hincapié en que esta expresión sólo da un límite superior para la capacidad de un canal ya que en la práctica hay que considerar otros factores; no obstante, indica adecuadamente la forma en cómo varia la tasa de datos máxima con el ancho de banda del canal y con el número de niveles de la señal digital transmitida a través de él.

EJEMPLO 18.5

Un canal sin ruido de 60 KHz nunca podrá transmitir señales binarias (n = 2) a una tasa de datos mayor de 120 Kbps; en efecto:

$$C = 2 \cdot AH \cdot \log_2(n) = 2 \cdot 60 \cdot \log_2 2 = 120\,Kbps$$

Tabla 18.3. *Ejemplos de tasas de datos de canales a través de distintos medios de transmisión.*

Tipo de medio	Medio	Velocidad	Rango de frecuencias	Casos o ejemplos	
Guiados	Pares trenzados	1 a 150 Mbps	0 a 1 MHz	• Canal telefono (4 KHz)	48 Kbps
				• Sistema T1	1,544 Mbps
				• Par trenzado sin blindaje (UTP)	1 a 100 Mbps
				• Par trenzado con blindaje (STP)	1 a 150 Mbps
				• RDSI	2 canales de 64 Kbps, y 1 de 16 Kbps
				• ADSL	512 Kbps/4 Mbps
				• Fast Ethernet	100 Mbps
				• Gigabit Ethernet	1 Gbps
	Cable coaxial	1 Mbps a 1 Gbps	0 a 500 MHz	• Ethernet	10 a 100 Mbps
	Fibra óptica	10 Mbps a 3,2 Tbps	180 a 370 THz	• Gigabit Ethernet	1 Gbps
				• Redes troncales (una señal)	OC-1: 52 Mbps a OC-192: 10 Gbps
				• Redes troncales con WDM	1.600 Gbps y superiores
				• ATM	2,5 Gbps
				• FDDI	2,5 Gbps
Inalámbricos	Radio	1 Mbps a 10 Gbps	30 MHz a 1 GHz	• Radio digital (banda 2 GHz)	1,5 a 45 Mbps
				• IEEE 802.11b/g (2,4/5 GHz)	5,5 a 11 Mbps
				• Espectro expandido (FHSS, 2,4 GHz)	1Mbps y 2 Mbps
			estación base: 869 a 894 MHz terminal móvil: 824 a 849 MHz	• Telefonía celular	9,6 a 19,2 Kbps
				• Radio convencional	1 a 10 Mbps
			1 GHz a 40 GHz	• Microondas	1 Mbps a 10 Gbps
			1 GHz a 100 GHz	• Satélites	1 Mbps a 10. Gbps (transponders de 50 Mbps)
	Infrarrojo	100 Kbps a 4 Mbps	0,3 a 428 THz	• IrDa	114 Kbs a 4 Mbs
				• Difusos (3,33 a 375 THz)	1 a 4 Mbps
				• Haz directo (3,33 a 375 THz)	1 a 10 Mbps

En el caso de canales con ruido, la capacidad de información del canal viene dada por la fórmula de Shannon-Hartley-Tuller:

[18.7]

$$C = AH \cdot log_2 \left(1 + \frac{S}{N}\right)$$

donde *S/N* es la relación señal/ruido.

EJEMPLO 18.6

Típicamente en un canal telefónico para transmisión de voz AH = 3.000 Hz, S = 0,1 mW y N = 0,4 μW. Por tanto, la relación señal-ruido es S/N = 250 ó $10 \cdot log_{10}(250) = 23{,}9794$ *dBs*
La capacidad de información del canal será:

$$C = 3.000 \cdot log_2 (1 + 250) \approx 24.000 \frac{b}{s}$$

18.2.3 Modulación

Para transmitir a grandes distancias (aproximadamente a más de 1,5 Km) es necesario efectuar una transformación en el mensaje denominada modulación. La **modulación** *permite transmitir señales a grandes distancias con atenuación muy baja* y utilizar el mismo canal para transferir varios mensajes simultáneamente, mediante un proceso electrónico denominado **multiplexación** (Sección 18.2.4). Con la multiplexación las señales van *mezcladas* por el canal y pueden separarse en los receptores por medio de una **demultiplexación** y/o **demodulación**, recuperándose los mensajes tal cual eran originalmente. Este es el caso, por ejemplo, de las emisoras y receptores de radio de una localidad. El canal, en este caso, es la atmósfera y en ella se encuentran simultáneamente las radiaciones emitidas por las distintas emisoras. Las emisoras modulan sus emisiones de forma tal que cada una de ellas ocupa frecuencias distintas. En los receptores se pueden seleccionar (*sintonizar*) las distintas bandas de frecuencia ocupadas por cada emisora específica. El receptor de radio efectúa la selección (o demultiplexación) y demodulación de la señal captada por la antena.

La modulación consiste en variar linealmente uno de los parámetros de otra señal denominada **portadora**, con la señal o mensaje a transmitir.

La portadora puede ser una señal sinusoidal (**modulación analógica**) o un tren de pulsos (**modulación digital**). De igual forma el mensaje puede ser analógico (voz o TV, por ejemplo) o digital (tal como datos a transmitir entre computadores).

En el caso de modulación analógica, los parámetros que se pueden modificar son los que caracterizan a la señal sinusoidal: amplitud o frecuencia o fase; teniendo en cada uno de los casos un sistema u otro de modulación: **Modulación de Amplitud (AM), Modulación de Frecuencia (FM)** o **Modulación de Fase (PM)**, respectivamente.

Para emitir una señal digital, y para mayor garantía de una adecuada transmisión, se suele hacer una codificación o modulación previa de dicha señal. Por lo general, para efectuar este proceso previo se parte (Figura 18.5a) de una señal binaria periódica, de período T y de duración de pulso τ (con $\tau < T$). En el caso de **Modulación por Anchura de Pulso** (**PDM** o **PWM**),

el grosor del pulso (τ) cambia según el valor del mensaje en ese momento (Figura18.5b). En la **Modulación por Posición de Pulso** (**PPM**) se varía la posición del pulso (τ) dentro del intervalo T (Figura 18.5c), y en la **Modulación Codificada por Pulsos** (**PCM**) el valor de la señal se recodifica en binario, con pulsos de igual duración y amplitud (Figura 18.5d).

Una vez que se tiene la señal digital en una de las formas anteriores (PDM, PPM o PCM) se puede emitir en una portadora analógica o digitalmente, según los casos que se indican en las Secciones 18.2.3.1 y 18.2.3.2.

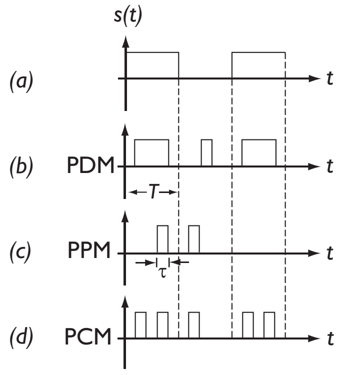


Figura 18.5. *Modulación de pulsos, (a) portadora; (b) modulación por anchura de pulso (PDM); (c) modulación por posición de pulso (PPM); (d) modulación codificada por pulsos (PPM).*

18.2.3.1 Transmisión analógica de una señal analógica

Es el caso clásico de transmisión realizado por la radio y televisión comerciales. La portadora es una señal sinusoidal y el mensaje es una señal que varía continuamente en el tiempo (audio, vídeo, etc.). En la Figura 18.6 se muestran ejemplos de modulación de amplitud, frecuencia y fase.

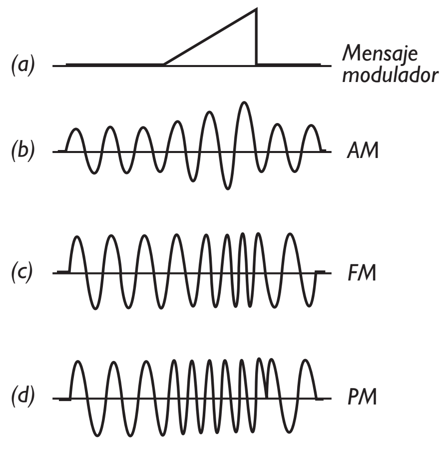


Figura 18.6. *Modulación de una señal sinusoidal con una señal triangular (a) Señal modulada; (b) Modulación en amplitud (AM); (c) modulación en frecuencia (FM); (d) Modulación de fase (PM).*

18.2.3.2 Transmisión analógica de una señal digital

Se suele utilizar uno de los cuatro siguientes tipos de modulación para efectuar físicamente la transmisión (Figura 18.7):

- **OOK** (*ON-Off-Keyed*), o **modulación binaria todo o nada** (Figura18.7a). La portadora se hace cero cuando el mensaje es cero.
- **ASK** *(Amplitud Shift Keying*) o **modulación por desplazamiento de amplitud** (Figura 18.7b).
- **FSK** (*Frequency Shift Keying)* o **modulación por desplazamiento de frecuencia**, Figura 18.7c. Se transmite una señal sinusoidal con una amplitud y fase constante. Durante la transmisión de un 1 se produce un cambio brusco o desplazamiento de la frecuencia de transmisión: al 0 le corresponde una frecuencia f_0, y al 1 otra frecuencia f_1. Por ejemplo, para una transmisión a 1200 bps estas frecuencias pueden ser f_0 = 2.100 Hz y f_1 = 1.300 Hz, y
- **PSK** *(Phase Shift Keying),* o **modulación por desplazamiento de fase** (Figura 18.7d). Al transmitir un 1 se produce, por ejemplo, un desfase de 180^0 en la portadora.

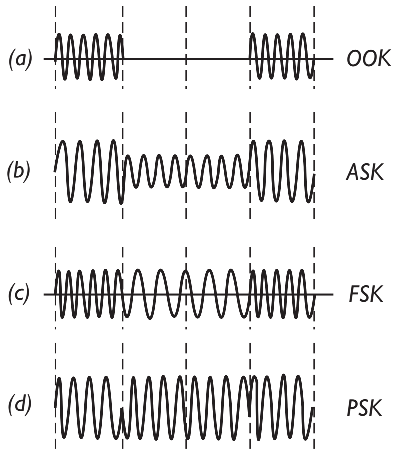


Figura 18.7. *Modulación de una señal sinusoidal con un mensaje digital (1001), (a) sistema OOK; (b) sistema ASK; (c) sistema FSK; (d) sistema PSK.*

Un **módem** *es un sistema para modular y demodular analógicamente señales digitales* (Figura 18.8). El objetivo fundamental de los módems es conseguir aprovechar los canales analógicos telefónicos de forma que pueda transmitirse por él información digital a la mayor velocidad posible.

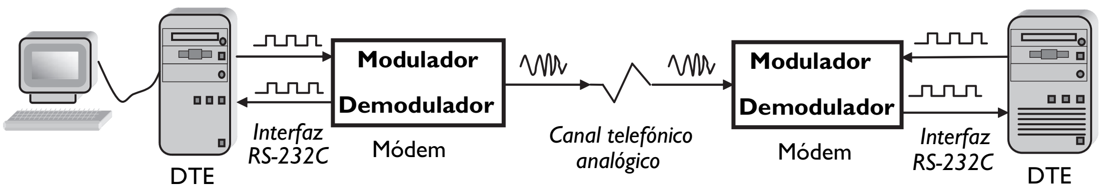


Figura 18.8. *Utilización de un módem para establecer una conexión de equipos informáticos a través de una red telefónica.*

Tradicionalmente los módems transmiten a una velocidad de 2.400 pulsos/s. Para aumentar la velocidad, en la señal sinusoidal portadora se incluyen varios niveles (de fase o de amplitud), en lugar de los dos correspondientes a un bit. De esta forma cada pulso (intervalo de tiempo) contiene más de un bit. Así, por ejemplo un módem que sigue el estándar V.26 de la CCITT utiliza modulación de fase (PSK). Se consideran 4 valores de fase (*tetrafase*), representando cada valor a dos bits sucesivos (a un **dibitio**); así se transmiten 2 bits/pulso Tabla 18.4. Este módem transmite a 1.200 pulsos/s, pero su tasa de datos es de 2.400 bps.

Con la transmisión en dibitios un módem de 1.600 baudios puede emitir a 3.200 bps.

En la Tabla 18.4b se indican los niveles del sistema DPSK en **octofase**, en el que en cada fase se transmiten tres bits (un **tribitio**); es decir, se transmiten 3 bits/baudio (recomendación V.27). Los módem de mayor velocidad (9600 baudios, recomendación V.29) utilizan **modulación de amplitud en cuadratura** o **QAM**. Este tipo de modulación combina modulación de fase (PSK) con modulación de amplitud. En el caso de la recomendación V.29 se consideran dos niveles de amplitud y 8 niveles de fase (octofase; es decir, se utilizan tribitios), con lo que se tienen 16 valores posibles de emisión (16-QAM). Se transmiten, por tanto, 4 bits/baudio. En estas circunstancias un módem de 2.400 pulsos/s emitirá a 9.600 bps.

Tabla 18.4. *Cambios de fase de emisiones de dibitios (a) y tribitios (b).*

Cambio de fase	Dibitio
0º	00
90º	01
180º	10
270º	11

(a)

Cambio de fase	Tribitio
0º	000
45º	001
90º	010
135º	011
180º	100
225º	101
270º	110
315º	111

(b)

Tabla 18.5. *Algunos estándares de modems.*

Estándar ITU-T	Velocidad de transferencia bps	Modulación	Pulsos
V.21	300	FSK	300
V.22	1.200	4-PSK	600
V.23	1.200	FSK	1.200
V.26	2.400	4-PSK	1.200
V.27	4.800	8-PSK	1.600
V.29	9.600	16-QAM	2.400
V.32 bis	14.400	64-QAM	2.400
V.33	14.400	128-QAM	2.400
V.34	28.800	4096-QAM	2.400
V.34 BIS	33.600	960-TCM	3.429
V90	56.000/33.600	TCM	
V92	56.000/48.000	TCM - 4 dimensiones	

Los módem de muy alta velocidad utilizan, además de los métodos descritos, la introducción de redundancias para detectar errores, denominándose este tipo de técnicas **modulación de código de rejilla** o **TCM** (*Trellis Coded Modulation)*. En la Tabla 18.5 se referencian algunos de los módems más utilizados.

18.2.3.3 Transmisión digital de una señal digital

En distancias cortas se puede transmitir la señal original, o bien directamente, o bien modulada en una de las formas indicadas anteriormente (PDM, PPM o PCM). En la emisión directa se dice que la transmisión se efectúa en **banda** o **modo base**.

Uno de los principales inconvenientes de transmitir la señal digital directamente, es que su espectro de energía está muy próximo a $f = 0$ (al nivel de continua) y los armónicos con frecuencias próximas a cero son muy atenuados, deformándose considerablemente la señal original. Para evitar esto se efectúa una transformación denominada **codificación de línea**. Estos códigos son idénticos a algunos de los utilizados para la grabación de información en soportes magnéticos (Sección 10.3.1). Concretamente se suelen utilizar los códigos NRZ (polar sin retorno a cero), BRZ (bipolar con vuelta a cero), PE (bifase o Mamchester), FM (difase diferencial) o MFM (Miller). Utilizando uno de estos códigos de línea la transmisión en modo base resulta más eficiente.

En la Figura 18.9 se representan los espectros de frecuencia de las señales codificadas según los procedimientos indicados anteriormente. En la figura, F representa el ancho de banda de la señal original, y f es la frecuencia del espectro de la señal ya codificada.

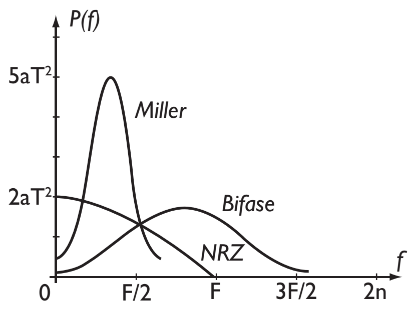


Figura 18.9. *Espectros de frecuencia obtenidos con distintos códigos de línea.*

18.2.3.4 Transmisión digital de una señal analógica

En primer lugar, la señal debe ser digitalizada por medio de un conversor A/D (Sección 11.5.1). La digitalización implica dos procesos: muestreo de la señal (Figura 18.10c) y codificación del valor de la muestra en binario. Este último proceso es, sencillamente, (Figura 18.10e) una modulación por codificación de pulsos (PCM).

El muestreo debe hacerse según indica el **criterio de Nyquist**. Este criterio establece [Car86, Leo01] que si se tiene una señal cuyo espectro sólo llega hasta una determinada frecuencia AF, puede ser transformada sin pérdida de información tomando muestras de la señal a una frecuencia (Figura 18.10b):

[18.8]

$$F_s = 2 \cdot AF$$

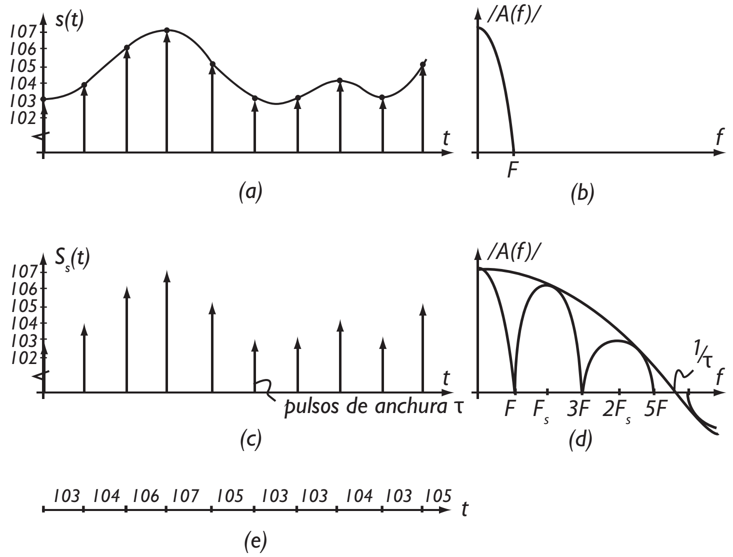


Figura 18.10. *(a) Señal s(t) a muestrear de ancho de banda F; (b) espectro de frecuencias de s(t); (c) muestras de anchura τ y período T_s=1/F_s; (d) espectro de las muestras (se supone que T_s = 5τ); (e) valores de las muestras a generar en binario.*

La limitación en frecuencia (a *AF*) de la señal original puede realizarse por medio de un circuito electrónico denominado **filtro pasa-baja**, que deja sólo pasar a su salida un rango o banda de frecuencias predeterminada de la señal de entrada. En este caso el ancho de la banda pasante de frecuencias debe coincidir con el ancho de banda *F*. El criterio de Nyquist, también conocido como **teorema del muestreo**, garantiza que con las muestras de una señal se puede reconstruir la señal original.

EJEMPLO 18.7

Si se desea transferir en forma digital una señal telefónica de voz filtrada a F = 4 KHz, es necesario muestrear dicha señal a F_s = 2 · F = 8 KHz. Esto quiere decir que las muestras se han de tomar cada T_s = 1/F_s = 125 µs; y se generan 8.000 muestras/segundo.
Si los valores de las muestras se transforman a binario (PCM), con un total de 8 bits (256 niveles de valores analógicos), la tasa de datos a la que habrá que transmitir la señal binaria será: 8 · 8000 = 64 Kbps.

La transmisión digital, frente a la analógica, tiene las siguientes ventajas:

a) Las técnicas y los circuitos digitales para tratamiento de señales están más perfeccionados y son más económicos que los analógicos.
b) Un pulso con ruido es más fácil de reconstruir que un valor analógico, permitiendo así transmisiones más fiables. En las estaciones repetidoras se incluyen circuitos restauradores de pulsos.
c) Permite la realización de multiplexación en el tiempo, obteniéndose un mejor aprovechamiento de los canales que con la multiplexación en frecuencia.

Debido a estas razones, las redes de comunicaciones públicas tradicionales (analógicas) se están transformando en digitales, incluso aunque las señales originales (audio, TV, etc.) sean analógicas. En las redes actuales, por tanto, coexiste la transmisión de señales de voz, datos e imágenes, codificadas o moduladas adecuadamente.

18.2.4 Multiplexación

Por lo general la capacidad de transmisión que ofrece un medio físico es mucho mayor que las necesidades de una comunicación entre dos estaciones. En este caso el medio puede ser compartido por diversas estaciones. La **multiplexación** *es una técnica que permite establecer múltiples transmisiones simultáneamente a través de un único medio o enlace.*

En este apartado consideraremos los tres procedimientos usados para multiplexar señales: multiplexación en frecuencia (Sección 18.2.4.1), multiplexación en el tiempo (Sección 18.2.4.2) y multiplexación de longitud de onda (Sección 18.2.4.3).

18.2.4.1 Multiplexación en frecuencia

En la Figura 18.11 se muestran los espectros de distintas señales y los espectros de las mismas, moduladas en amplitud con distintas frecuencias portadoras (f_{ci}). Como se aprecia, al modular en amplitud el espectro aparece desplazado según determina la frecuencia de la señal portadora. (Los espectros que aparecen en la Figura 18.11 corresponden a un tipo de modulación denominado Modulación de Amplitud en Banda Lateral Única o AM-SSB) [Car86].

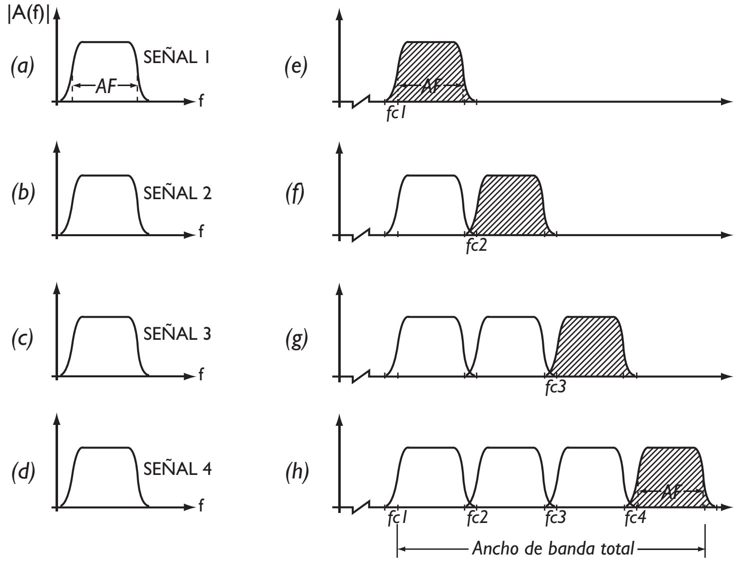


Figura 18.11. *Multiplexación en frecuencia de distintas señales moduladas en amplitud.*

Si la banda pasante del canal lo permite (incluye el rango de frecuencias f_{c1} a $f_{c4} + AF$) se pueden emitir por el mismo canal las cuatro señales sin que sus espectros se solapen (Fig. 18.10h). El receptor se encargaría de seleccionar cada una de las señales, tomando como referencia cada una de las frecuencias portadoras (f_{ci}). De esta forma se transmiten entre

centrales telefónicas analógicas los denominados grupos maestros. Un **grupo maestro** esta constituido por 600 canales de voz con portadoras separadas en 4 KHz, y ocupando un ancho de banda total de 2.400 KHz (de 60 a 2.540 KHz).

Este proceso se denomina multiplexación por división de frecuencia (o abreviadamente, multiplexación de frecuencia). La **multiplexación por división en frecuencia** permite transmitir por un mismo medio distintos mensajes, trasladando el espectro de frecuencias de cada uno de los mensajes por modulación y de forma que los espectros no se solapen.

La multiplexación de frecuencia se utiliza en prácticamente todos los casos de modulación con portadora analógica. En radiocomunicaciones cada emisora de radio tiene asociados una frecuencia de portadora y un ancho de banda. Evidentemente el número de señales que pueden emitirse simultáneamente por un medio determinado depende del rango de frecuencias que no atenúa el medio (**banda pasante**) y del ancho de banda de los mensajes.

18.2.4.2 Multiplexación en longitud de onda

La multiplexación en longitud de onda o **WDM** (WDM, *Wavelenght-division multiplexing*) se utiliza en transmisiones ópticas y es exactamente lo mismo que la modulación de frecuencia, pero en el rango del espectro óptico.

Una fibra óptica tiene una capacidad de comunicaciones muy elevada ya que dispone de dos anchos de banda aprovechables (de muy baja atenuación), cada uno de ellos de 18 y 19 THz, respectivamente[1]. Estas bandas pasantes se podrían utilizar para miles de canales de transmisión de decenas de Gbps, que es el rango en que funcionan los sistemas de transmisión óptica y el límite de funcionamiento de la tecnología electrónica.

El fundamento de la WDM consiste en modular cada canal de transmisión a una frecuencia determinada (f_i) o *color* (recuérdese que cada frecuencia del espectro visible corresponde a un color determinado), obteniéndose así distintos haces ópticos centrados en distintos colores o longitudes de onda[2]. La mezcla o multiplexación, así como la separación o demultiplexación, sencillamente se realiza utilizando prismas (Figura 18.12) o rejillas de difracción.

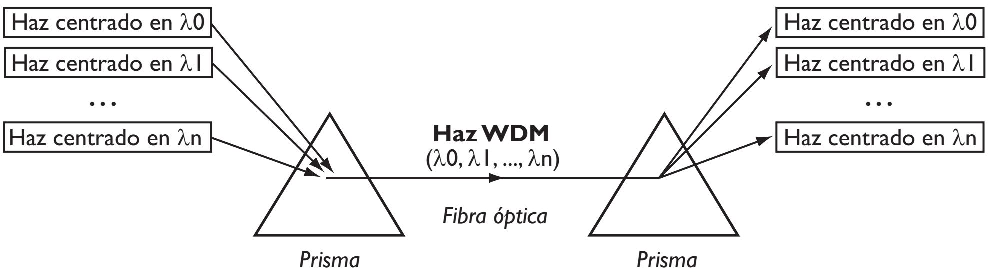


Figura 18.12. *Multiplexación y demultiplexación en longitud de onda utilizando prismas.*

En la práctica hay sistemas WDM que multiplexan 32 señales ópticas de 10 Gbps (OC-192) obteniéndose en total un haz que transporta información a una velocidad de 320 Gbps.

[1] THz significa Teraherzio y 1 THz = 10^9 Hz = 1.000 GHz.

[2] Recuérdese que la longitud de onda, λ, es inversamente proporcional a la frecuencia, f, $\lambda = v/f$, donde v es la velocidad de propagación en el medio; en el espacio libre $v = c = 3 \cdot 10^8$ m/s, que es la velocidad de la luz.

18.2.4.3 Multiplexación en el tiempo

Con las transmisiones digitales puede utilizarse **Multiplexación por División en el Tiempo (TDM)**. Este tipo de multiplexación aprovecha los tiempos inactivos que hay entre pulso y pulso de la señal digital o muestreada (Figura 18.10c). Si, por ejemplo, se transmite una señal de ancho de banda AF = 3 KHz muestreada con un período $T_s = 1/(2AF) = 167$ μs y ancho de pulso $\tau = 5$ μs, por cada período de tiempo T_s hay un intervalo de $t = 167 - 5 = 162$ μs desaprovechado. En este intervalo de tiempo podríamos incluir, por ejemplo, otros 15 pulsos de 5 μs correspondientes a otras tantas señales de 3 KHz de ancho de banda. Estas señales deben transmitirse siempre en el mismo orden *S1, S2,..., S16, S1, S2,...,S16, S1,...,* etc. (Figura 18.13), de esta forma todas ellas tienen la misma frecuencia Fs = 6 KHz y se transmiten, sencillamente, desfasando cada una de ellas de la anterior en 10 μs. Entre la muestra S16 y S1 se incluye un desfase de 17 μs, marcando con ello el inicio de una nueva serie de 16 muestras de cada una de las señales (una nueva **trama**).

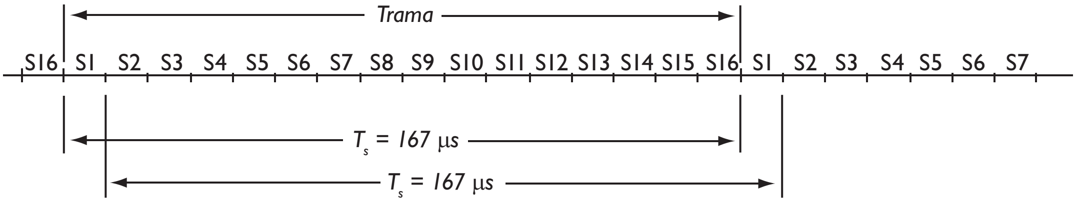


Figura 18.13. *Orden en que deben transmitirse las muestras (S) de 16 señales multiplexadas en el tiempo.*

EJEMPLO 18.8

El sistema T1 para telefonía desarrollado por Bell System utiliza una multiplexación TDM para transmitir simultáneamente 24 canales de voz de 4 KHz (Figura 18.14). Cada señal está modulada en PCM/ASK, de forma que una muestra analógica se transforma en 8 bits. Cada trama está formada por 24 · 8 = 192 pulsos más uno adicional de sincronismo (193 pulsos en total). A continuación se obtiene la frecuencia de transmisión de los pulsos.

Como las señales tienen un ancho de banda de AF = 4 KHz, para no perder información cada una de ellas debe muestrearse a Fs = 4 · 2 = 8 KHz; es decir, Ts = 1/Fs = 125 μs. Como cada trama ocupa 125 μs y tiene 193 pulsos, el período entre pulsos será: T = 125/193 = 0,648 μs; es decir, la frecuencia es: f = 1/T = 1.544 Mbps.

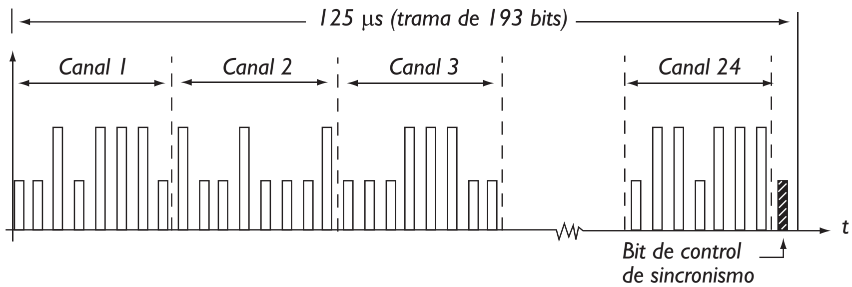


Figura 18.14. *Sistema TDM para transmisión de 24 canales de voz.*

18.2.5 Conexiones serie

Las conexiones internas de un computador suelen ser de tipo **paralelo**, de forma que se dispone de tantas líneas de interconexión como número de bits tenga cada dato o palabra a transmitir. En redes de computadores las conexiones se hacen en modo **serie**, para abaratar los costes ya que sino habría que contratar una línea de comunicaciones para cada bit de palabra. Las conexiones serie se suelen realizar de dos formas: o asíncrona o síncrona, según se detalla a continuación.

18.2.5.1 Conexiones asíncronas

En los enlaces asíncronos los datos sucesivos aparecen en tiempos arbitrarios y no existe un control de reloj especificando un período de tiempo entre dato y dato. Por lo general cada dato es un carácter de 8 bits, y es tratado como unidad de transmisión. Entre la emisión de un carácter y el siguiente no existe un intervalo de tiempo predeterminado, transmitiéndose los caracteres asíncronamente. Los bits sucesivos que componen el carácter, sin embargo, se transmiten síncronamente ya que los bits sucesivos internos al carácter se emiten según una frecuencia preestablecida. A cada carácter se le añaden unos bits de control, que identifican el principio y el fin del carácter, y un bit de paridad para control de errores. El receptor detecta la llegada y fin del carácter y comprueba la paridad, corrigiéndose el error, en su caso.

Por lo general cada byte se transmite con los siguientes bits (cada bit dura un determinado tiempo, T):

- Bit de comienzo: 1 bit (bit *0*).
- Bits del dato: 5, 6, 7, u 8 bits, según el código utilizado para transmitir la palabra-dato o byte-dato.
- Bit de paridad: par o impar. A veces no se utiliza el bit de paridad o, sencillamente, se dedica un bit a ella pero no se comprueba.
- Bit(s) final o de parada: 1 bit o 1 1/2 bits ó 2 bits (1 1/2 bits quiere decir una duración $3 \cdot T/2$) (bit *1*).

EJEMPLO 18.9

En la Figura 18.15 se muestra cómo se transmitirían los caracteres ASCII 5V, en un enlace serie asíncrono con los siguientes parámetros:

bits de final: 1 1/2
bits de datos: 7, código ASCII (ver Sección 4.2.2)
bit de paridad: par

Con la transmisión asíncrona se pueden transmitir grupos de caracteres (un carácter o varios) no necesariamente de la misma longitud. Cuando hay conexión pero no se envía ninguna palabra, la línea permanece en el *nivel activo* (nivel alto); este nivel también se denomina *marca*. La llegada de un carácter (en cualquier instante) se detecta porque la señal de la línea cae durante un intervalo de tiempo T (llegada del bit de comienzo) al nivel de reposo (también llamado *espacio*). El momento de detección de un carácter se sincroniza con el reloj del receptor, que, por cuestiones de precisión, es de una frecuencia mayor (8, 16 ó 32) que la del reloj del emisor.

En cuanto a las interfases, tradicionalmente se vienen utilizando la RS-232-C (también denominada V.24) y la RS-422. Las denominaciones RS (iniciales de *Recomended Standard*) han sido propuestas por la EIA (*Electronic Industries Association*). Por otra parte las interfases *X* y *V* están definidas por el CCITT (Sección 19.1.1).

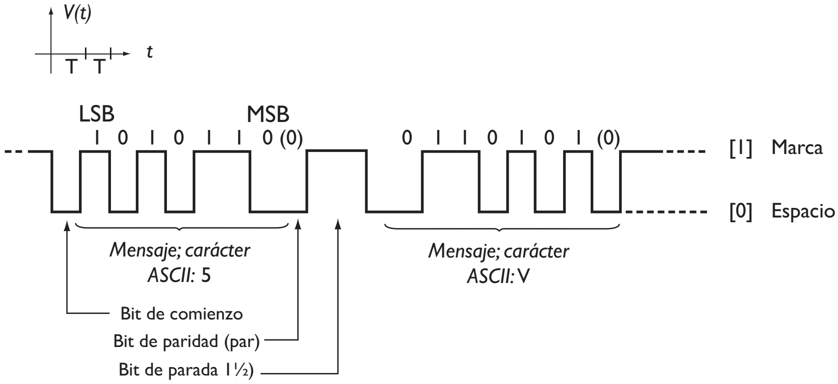


Figura 18.15. *Transmisión serie de los caracteres ASCII "5V", en modo asíncrono.*

18.2.5.2 Conexiones síncronas

Los datos sucesivos se emiten en intervalos de tiempo específicos, controlados por un reloj. De esta forma no sólo se envían sincronizados los bits, sino también los caracteres. Estos se transmiten en grupos o bloques, y la información de control para sincronizar el receptor con el emisor no se introduce en cada carácter transmitido, sino en forma de caracteres especiales de sincronización o de control de errores (paridad horizontal y CRC, Sección 18.2.6). En transmisiones ASCII se utilizan para sincronizar caracteres **SYN** (1001 011(0)). Estos caracteres sirven para ajustar con precisión el reloj del receptor, que deberá captar adecuadamente los caracteres de información que le siguen (Figura 18.16). Dentro de los bloques de información se suelen incluir caracteres **SYN**. Así, si en el receptor se detecta un error de transmisión (error de paridad en uno de los caracteres, por ejemplo), se puede repetir la transmisión desde el último carácter **SYN** recibido y no obligatoriamente desde el comienzo del bloque.

En las transmisiones síncronas, a diferencia de las asíncronas, se controlan bloques de información (cientos o miles de bytes) en lugar de caracteres. Se elimina la necesidad de los bits de comienzo y final incluidos en cada carácter, consiguiéndose una velocidad efectiva mayor de comunicación de datos (caracteres usuario/segundo). Los controles de errores y redundancias se incluyen considerando al bloque en su totalidad consiguiéndose una mayor calidad en la transmisión. Como consecuencia permiten velocidades mayores (bps) a mayores distancias.

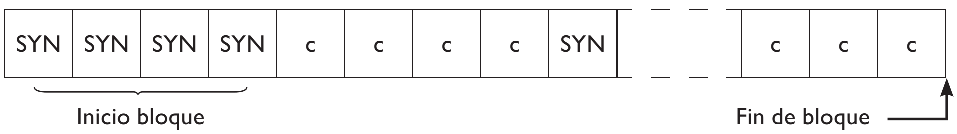


Figura 18.16. *Transmisión síncrona de un bloque de caracteres ASCII (c representa a un carácter ASCII de 8 bits).*

Tradicionalmente en transmisiones síncronas se utilizaba el protocolo *BInary SYNCrhonous communication* o, abreviadamente, **BISYNC**. En la actualidad, además del protocolo BISYNC se utilizan otros, como el **DCMP**, (desarrollado por DEC) o el **HDLC**. En comunicaciones síncronas se pueden utilizar algunas de las interfaces ya vistas (tal como la RS-422). En la actualidad es muy común utilizar la interfaz X.21 del CCITT, que corresponde al nivel físico de la recomendación X.25 (ver, por ejemplo, [Sta04]).

18.2.6 Control de errores en comunicaciones digitales

En comunicaciones digitales a un conjunto predeterminado de bits consecutivos lo denominamos **unidad de datos**. Antes de transmitir una unidad de datos se le añaden, en el receptor, redundancias para que el receptor pueda comprobar si durante la transmisión se han producido errores (ruido ambiental, interferencias, etc.). Los errores pueden ser de los siguientes tipos (Tabla 18.6):

- **Errores en bits aislados.**
- **Errores en bits múltiples,** cuando dos o más bits no consecutivos en una secuencia dada (unidad de datos) cambian.
- **Errores en ráfagas,** cuando dos o más bits consecutivos en una unidad de datos cambian.

Tabla 18.6. *Errores producidos en la transmisión y formas de detectarlos.*

Tipos de errores	– Errores en bits aislados. – Errores en bits múltiples. – Errores en ráfagas.
Métodos de detección de errores	– Comprobación de redundancia vertical (VCR, *Vertical Redundancy Checking*). – Comprobación de redundancia horizontal (LCR, *Longitudinal Redundancy Checking*). – Suma de comprobación (*Checksum*). – Redundancias cíclicas (CRC, *Cyclic Redundancy Checking*).

La inclusión de redundancias para control de errores suele realizarse añadiendo al final de la unidad de datos una información calculada en función de su contenido. Esta información adicional se produce en el emisor. Al llegar la unidad de datos junto con los bits de control de error al receptor, éste recalcula los bits de control y comprueba si coinciden con los bits de error recibidos. Si no es así, claramente ha habido un error en la transmisión, hecho que el receptor comunica al emisor para que le retransmita nuevamente la unidad de datos junto con sus bits de control de error.

En esta sección se dan unas ideas sobre los procedimientos más usados para calcular los bits de comprobación de errores (Tabla 18.6). Hay otros métodos, y lo que se pretende con todos ellos es detectar el mayor número posible de errores.

La forma más sencilla de detectar errores consiste en incluir bits de paridad (Sección 3.7). Usualmente lo que se hace es añadir al final de cada unidad de datos un bit, de tal forma que el número total de unos de la unidad transmitida (unidad de datos original junto con su bit de paridad) sea par (o impar). El receptor comprueba si el número de bits uno es par (o impar), y en el caso de que no lo sea ordena que se le retransmita el carácter. Este procedimiento

se denomina **bit de paridad de redundancia vertical, VCR**. Este sistema, como se indicó en la Sección 3.7, tiene el inconveniente de que no se pueden detectar errores cuando se producen en un número par de bits (cuando a causa de errores de transmisión se alteran 2, 4, 6,... bits). Para mejorar la detección de errores, se suele añadir un **bit de paridad de redundancia horizontal**, **LCR**. Con este procedimiento (Figura 18.17) cada cierto número k de unidades de datos ($k = 8$, en el ejemplo de la figura) se añade un carácter de comprobación. El bit menos significativo de la unidad de comprobación corresponde a la paridad de los k bits menos significativos de las unidades del mensaje, el bit de orden uno del carácter de comprobación es el bit de paridad correspondiente a los k bits de orden uno de las unidades de datos, y así sucesivamente. Recordar que, según se vio en el Capítulo 4, el bit de paridad (criterio par) se obtiene sin más al hacer la operación exclusive-OR (o suma módulo 2) de los bits correspondientes.

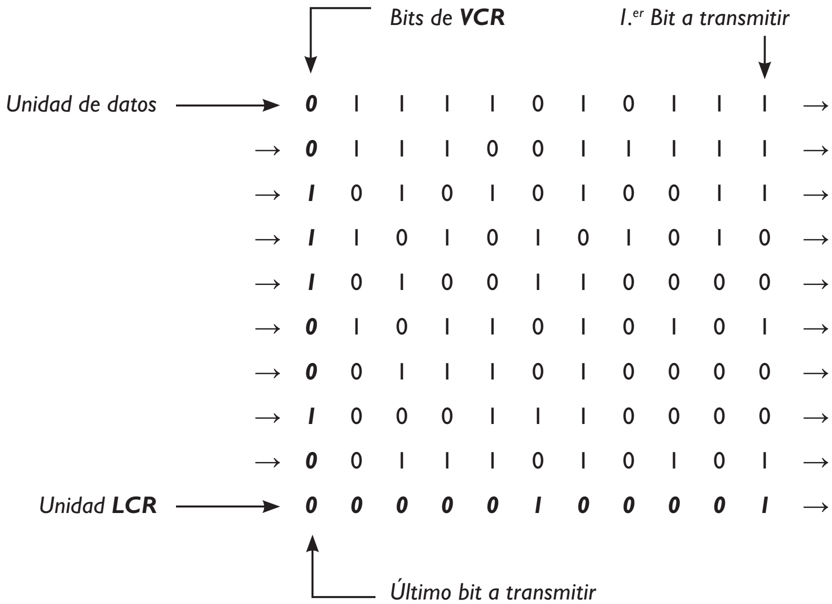


Figura 18.17. *Bits de paridad de redundancia vertical (VCR) y de redundancia horizontal (LCR). (Las flechas horizontales indican el orden en que se transmiten los bits).*

Otro método de detección de errores muy usado en transmisiones es el de la **suma de comprobación**. Como en el caso de bit de paridad de redundancia horizontal, consiste en dividir la unidad de datos (de m bits), en n bloques de k bits. Las redundancias a incluir se forman sumando los k bloques en complemento a uno, y complementando el resultado final. Los k bits así obtenidos, S_E, se incluyen al final de los m bits de la unidad de datos original. En el receptor, cuando llega una nueva unidad de datos completa, se extraen los primeros n bloques y se efectúa su suma, S_E. Posteriormente se suman la S_E recibida con la S_R calculada y, si la transmisión ha sido correcta, el resultado será cero.

Cuando se producen errores a ráfagas (cambian varios bits seguidos), los procedimientos de detección anteriores son prácticamente inútiles. Uno de los procedimientos más usados consiste en utilizar códificación **CRC** procedimiento también denominado de **codificación polinómica**.

Como en los casos anteriores, las unidad de datos se divide en bloques sucesivos de k bits. A los k bits del mensaje se le añaden r bits de comprobación de error, obtenidos según el procedimiento que se describe a continuación.

Los k bits de cada bloque se tratan como si fuesen los coeficientes de un polinomio, $M(x)$, de orden $k - 1$, en el que las operaciones se hacen en módulo-2. Así, por ejemplo, si el mensaje fuese (para simplificar supóngase $k = 8$):

10010110

el polinomio considerado sería:

[18.9]

$$M(x) \equiv 1 \cdot x^7 + 0 \cdot x^6 + 0 \cdot x^5 + 1 \cdot x^4 + 0 \cdot x^3 + 1 \cdot x^2 + 1 \cdot x^1 + 0 \cdot x^0 = x^7 + x^4 + x^2 + x$$

Por otra parte se utiliza un **polinomio generador**, $G(x)$ de grado r. Este polinomio está predeterminado, y es el mismo en el emisor y el receptor. Los polinomios normalizados que más se suelen utilizar (para $k = 16$) son:

[18.10]

$$\text{CRC-16} \rightarrow G(x) \equiv x^{16} + x^{15} + x^2 + 1$$

[18.11]

$$\text{CRC-CCITT} \rightarrow G(x) \equiv x^{16} + x^{12} + x^5 + 1$$

El objetivo del procedimiento consiste en añadir r bits al mensaje de k bits, de forma tal que el polinomio, $T(x)$ correspondiente a los $k + r$ bits sea divisible por $G(x)$. El receptor verifica si $T(x)$ es divisible por $G(x)$, en caso de no serlo hay un error en la transmisión.

El algoritmo es el siguiente:

a) Añadir r bits ceros al extremo de menor orden del mensaje. El polinomio correspondiente será: $x^r \cdot M(x)$

b) Dividir (módulo 2) $x^r \cdot M(x)$ entre G(x):

[18.12]

$$\frac{x^r \cdot M(x)}{G(x)} = C(x) + \frac{R(x)}{G(x)}$$

Donde $C(x)$ es el cociente y $R(x)$ es el resto. $R(x)$ siempre es menor que $G(x)$, es decir, será de grado r o menor.

c) Restar a $x^r \cdot M(X0)$ el valor del resto $R(x)$:

[18.13]

$$T(x) \equiv x^r \cdot M(x) - R(x)$$

Se obtiene así $T(x)$, cuyos coeficientes (unos o ceros) constituyen el mensaje a transmitir. $T(x)$ es siempre divisible por G(x), ya que siempre que se reste al dividendo el resto, lo que resulta es divisible por el divisor. En efecto, de [18.12] y [18.13]:

[18.14]

$$\frac{T(x)}{G(x)} = \frac{x^r \cdot M(x) - R(x)}{G(x)} = C(x)$$

el resto resulta ser cero.

El emisor efectúa la división de *T(x)* por *G(x),* y si el resto no es cero hay error de transmisión.

En el algoritmo anterior, según se indicó anteriormente, las operaciones se realizan en módulo-2. Esto facilita la implementación hardware del algoritmo, ya que en las operaciones no hay que considerar acarreos ni débitos. La suma y resta en módulo-2 se reduce a efectuar la operación exclusive-OR bit a bit, y en la división módulo-2 las restas se efectúan módulo-2.

Se demuestra que con este procedimiento, utilizando el polinomio generador CRC-CITT, Expresión [18.11], se detectan:

1. Todos los errores de 1 bit individual.
2. Todos los errores que alteren dos bits.
3. Todos los errores que alteren un número impar de bits.
4. Todos los errores de ráfagas de 16 ó menos bits.
5. El 99.997% de errores en ráfagas de 17 bits.
6. El 99.998% de errores en ráfagas de 18 ó más bits.

Para más detalles pueden verse, por ejemplo, las referencias bibliográficas [Tan03, Sta05 o Leo01].

18.3 Conclusión

Los conceptos estudiados en este capítulo sirven de base para entender adecuadamente los contenidos del siguiente capítulo, dedicado a las redes de computadores e Internet.

En primer lugar hemos descrito los objetivos que se pretenden cubrir con las redes de comunicación (Sección 18.1). Posteriormente (Secciones 18.2.1 y 18.2.2) hemos tratado sobre la naturaleza de las señales que transportan la información y de los medios a través de los cuales se hace dicho transporte. A la transformación que se realiza en las señales portadoras para insertar en ellas los mensajes, proceso denominado modulación, se ha dedicado la Sección 18.2.3. La multiplexación, o técnica con la que se logra aprovechar con gran eficiencia la capacidad de un canal de comunicación logrando transmitir simultáneamente diversas señales, se ha discutido en la Sección 18.2.4. El estudio de las comunicaciones serie, forma bajo la que se realizan las transmisiones de datos a distancias no muy cortas (mayores de unos pocos metros) se ha realizado en la Sección 18.2.5. El capítulo finaliza (Sección 18.2.6) presentando algunos algoritmos utilizados para el control de errores en comunicaciones digitales.

EJERCICIOS

18.1 Obtener la tasa de datos requerida para una señal de vídeo con resolución SVGA y con una cadencia de 30 cuadros/segundo, suponiendo que cada píxel se codifica con 8 bits y que no hay compresión de datos.

18.2 Se desea transmitir en banda base la siguiente información:

1010 0011 1010

¿Cuál sería la señal a transmitir codificada en NRZ, BRZ, PE, FM, y MFM?

¿Con cual de los códigos anteriores se necesitaría un ancho de banda de canal menor? Razone la respuesta.

18.3 Hacer una estimación (con las hipótesis que se estime oportuno) del número de conversaciones telefónicas que se podrían transmitir por un canal de 64 KHz,

a) Utilizando multiplexación en frecuencia.

b) Utilizando multiplexación en el tiempo.

Suponer que se hace en PCM, con 6 bits por muestra, y que cada muestra ocupa $T = 0{,}5$ µs.

18.4 Un sistema de adquisición de datos analógicos de 16 canales se utiliza para captar otras tantas señales, cada una de las cuales es de un ancho de banda de B = 1 KHz. Si la frecuencia de muestreo (para no perder información) debe ser al menos de f_s = 2KHz. cuál es el tiempo máximo de conversión posible de cada muestra para el conversor A/D?

18.5 Obtener los bits de paridad (criterio impar) de redundancia vertical (VCR) y de redundancia horizontal (LCR) del siguiente mensaje, suponiendo que las unidades de datos son de 16 bits:

1011 1011 1011 0101 0100 0001 0111 1111 1001 0000 0101 1001 0111 0101 0101 0111

18.6 Obtener la suma de comprobación del mensaje del ejercicio anterior.

18.7 Obtener el código de redundancia cíclica (CRC) para el siguiente bloque de bits:

1010 0110 0111 1000

utilizando el polinomio generador CRC-CCITT.

BIBLIOGRAFÍA

[Car02] Carlson, A. B.; Crilly, P. B.; Rutledge, J. C.: *Communication Systems,* 4ª Edc. McGraw-Hill, 2002.

[For01] Forouzan, B.: *Transmisión de datos y redes de comunicaciones*, 2ª Edc. McGraw-Hill, 2001.

[For00] Forouzan, B.: *TCP/IP. Protocol Suite*, McGraw-Hill, 2000.

[For03] Forouzan, B.: *Local Area Networks*, McGraw-Hill, 2003.

[Gar03] García, P.; Díaz, J.; López, J. M.: *Transmisión de datos y redes de computadores*, Prentice Hall, 2003.

[Hay94] Haykin, S.: *Communication Systems*, John Wiley & Sons, 1994.

[Kur01] Kurose, J. F.; Ross, K. W.: *Computer Networking,* Addison Wesley, 2001.

[Leo01] León-García, A.; Widjaja, I.: *Redes de comunicación. Conceptos fundamentales y arquitecturas básicas,* MacGraw-Hill, 2001.

[Sta04] Stallings, W.: *Comunicaciones y redes de computadores*, 7ª Edc. Prentice Hall, 2004.

[Tan03] Tanenbaum, A. S.: *Redes de computadoras,* 4ª Edc. Prentice Hall, 2003.

[Whi04] White, C. M.: *Data communications & Computer Networks*, Thomson, 2004.

CAPÍTULO 19

Redes de computadores e Internet

Las redes de computadores representan una de las tecnologías que han sufrido mayor crecimiento en los últimos años. Este desarrollo es debido al gran numero de aplicaciones que se utilizan a través de las redes, entre las que se pueden incluir desde las clásicas de utilización remota de recursos de cómputo, correo electrónico, transferencia de archivos y acceso a páginas web, hasta las más novedosas de acceso a radio y TV, comercio electrónico, vídeo bajo demanda, telefonía fija y móvil, videoconferencia, y, en general, **aplicaciones multimedia** bajo diversas formas. Por otra parte, poco a poco las redes de computadores se van imbricando más con la arquitectura de computadores, de forma tal que se ha producido un solapamiento notable entre el *procesamiento* de datos y las *comunicaciones* de datos, dando lugar a una mayor integración entre los sistemas que *procesan* información y los que la *transmiten*.

La primera parte de este capítulo (Sección 19.1) pretende dar una visión general sobre las redes de computadores; para ello, en primer lugar se presentan distintas topologías de interconexión (Sección 19.1.2), para, seguidamente, describir los conceptos básicos del modelo de referencia OSI que establece un marco sobre la forma en la que sistemas diferentes cualesquiera pueden comunicarse entre sí (Sección 19.1.3). Las tres secciones siguientes se dedican a los tres tipos de redes más relevantes: las de área personal o PAN (Sección 19.1.4), las de área local o LAN (Sección 19.1.5) y las de área amplia o WAN (Sección 19.1.6).

La Sección 19.2 se consagra íntegramente a Internet. La información que se transmite a través de una red, o entre redes, es controlada y dirigida por medio de distintos dispositivos o procesadores de red, que se describen el la Sección 19.2.1. Posteriormente se explica cómo se realiza el direccionamiento en Internet (Sección 19.2.2), seguidamente se describe el conjunto de protocolos TCP/IP (Sección 19.2.3) que es el que ha permitido el desarrollo de Internet. La Sección 19.2 concluye con una breve descripción de las aplicaciones básicas de Internet (Sección 19.2.4). El último apartado de este capítulo se reserva a la World Wide Web (www), conocida abreviadamente por web (Sección 19.3), considerando aspectos tales como el direccionamiento en la web (Sección 19.3.1), el protocolo http (Sección 19.3.2) y el concepto de navegador web (Sección 19.3.3). El capítulo concluye con unas reflexiones finales (Sección 19.4).

19.1 Redes de computadores

La conexión de equipos informáticos a través de redes de comunicaciones se remonta al sistema SAGE (*Semi-Automatic Ground Enviroment*) desarrollado a partir de 1956 como parte del esquema de defensa aérea de los Estados Unidos, formado por redes de 23 computadores. Cada red conectaba los radares de cada zona a un computador central [Leo01]. A partir de entonces se fueron desarrollando distintas redes que se caracterizaban por disponer de un gran computador central (*mainframe*) al que se interconectaban terminales remotos. Estos terminales eran sencillos periféricos como lectores de tarjetas, impresoras, perforadores de tarjetas y terminales interactivos teclado/pantalla, que se conectaban a la red analógica de telefonía convencional (de voz) por medio de módems. Las redes descritas se denominaban **redes orientadas a terminales**. Los computadores centrales de estas redes disponían de sistemas operativos de multiprogramación, con funcionamiento en modo de colas series (*bach*) y conversacional (tiempo compartido). Un ejemplo de este tipo de sistemas lo constituye la red creada a partir de 1969 por el Centro de Proceso de Datos del Ministerio de Educación y Ciencia de España, que estaba formada por un computador Univac 1100, a la que se conectaban distintos terminales ubicados en el propio centro y en 21 universidades o centros de investigación distribuidos a lo ancho de la geografía española[1]. Las líneas telefónicas eran contratadas por horas, y esta red hizo posible a numerosos investigadores acceder a grandes recursos de cómputo con costes locales muy reducidos.

En el mismo año que se inició la red española citada anteriormente (1969), comenzó en Estados Unidos la construcción de una red de computadores en la que los elementos a conectar eran computadores y no simples terminales (*terminales tontos*). Este proyecto se denominó ARPANET (*Advanced Research Projects Agency Network*) y fue financiado por el Ministerio de Defensa de los Estados Unidos. El objetivo era construir una red de computadores de forma tal que, si por un ataque enemigo, uno de los computadores o un enlace eran destruidos, el sistema globalmente siguiese funcionando. Una de las ideas básicas consistía en que el computador de origen dividía los mensajes en unidades denominadas **paquetes**, que viajaban independientemente uno de otro, e incluso por caminos distintos, a través de los nudos de la red, hacia el computador destino. ARPANET desarrolló los principales protocolos de comunicaciones que se utilizan en la actualidad para Internet, por lo que puede considerarse su precursora.

En la década de los años 80, en otro orden de cosas, se inició un descenso espectacular del precio de los computadores, fundamentalmente bajo la forma de sistemas PC, y, como consecuencia, se produjo un incremento de su uso. Estos equipos podían fácilmente conectarse con módems a la red telefónica y, con un pequeño emulador de un *terminal* intercambiar información con cualquier computador conectado a dicha red. De esta forma los *terminales tontos* fueron sustituyéndose por *terminales inteligentes,* distribuyéndose el procesamiento de las aplicaciones y el almacenamiento de la información entre los distintos equipos interconectados.

Hoy en día la mayoría de redes están interconectadas unas con otras formando una red mundial de computadores, que se conoce con el nombre de Internet. Internet está constituida por equipos de muy diversa capacidad (desde supercomputadores a computadores móviles), y proporciona a decenas de millones de personas la utilización de múltiples servicios como acceso a recursos hardware, correo electrónico, transferencia de archivos, noticias, acceso a documentos multimedia, transacciones bancarias y comercio electrónico.

[1] El equipo central, después de sucesivas ampliaciones, llegó a estar valorado, en 1975, en unos dos millones y medio de euros.

Una **red de computadores** *es un conjunto de computadores autónomos interconectados a través de un medio (hilos de cobre, cable coaxial, microondas, satélites, fibras de vidrio, etc.) por el que intercambian información.*

Todo computador, para conectarse a una red, necesita un hardware específico, **módem** o **tarjeta de interfaz a la red** (NIC), y un programa especial de comunicaciones. Ambos elementos deben adecuarse a las características y protocolos de la red.

No debe confundirse una red de computadores con un sistema informático distribuido. Un **sistema distribuido** *es una red de computadores que dispone de un sistema operativo o software potente de forma tal que desde cualquier máquina se puedan solicitar los servicios o aplicaciones por su nombre, sin necesidad de que previamente el usuario tenga que requerir o abrir explícitamente una sesión con el computador respectivo*. En otras palabras, en un sistema distribuido el usuario ve al sistema como si fuese un único computador, concepto que suele denominarse **SSI** (*Single System Image*).

Las redes de computadores consiguen aumentar el cociente prestaciones/precio de sistemas informáticos, obteniéndose entre otras, las siguientes ventajas:

- Posibilitan un servicio remoto de utilización de aplicaciones, sin necesidad de que el usuario disponga realmente de ellas.
- Pueden establecerse procesos en distintos computadores que actúen en paralelo, colaborando para la realización de una determinada tarea.
- Permite el acceso a documentos de tipo texto o multimedia, en general. Con estos servicios se puede acceder a información de bases de datos y archivos desde computadores ubicados a grandes distancias.
- Admiten la gestión de bases de datos distribuidas, de forma que un único sistema no necesite tener capacidad suficiente para albergar la base de datos completa.
- Aumentan la seguridad del sistema a través de la redundancia (si un computador de la red está averiada se puede utilizar otro de la red).
- Si el equipo local, al que se tiene acceso directo, no dispone de las prestaciones adecuadas (poca memoria, está muy cargado de trabajo, no dispone de impresoras rápidas o de suficiente calidad de impresión, etc.), el usuario puede conectarse a otro equipo de la red que reúna las características pertinentes.
- Posibilitan la existencia de sistemas de control industrial constituidos, por ejemplo, por varios computadores de uso específico de una fábrica, interconectadas entre sí.
- Permiten la utilización de la red de computadores como medio de comunicación: correo electrónico, radio y televisión a través de internet, etc.

Este capítulo se inicia (Sección 19.1.1) con una referencia a las dos organizaciones más relevantes que establecen normalizaciones sobre redes de comunicaciones. Después analizaremos distintas topologías de redes (Sección 19.1.2), para presentar a continuación (Sección 19.1.3) los fundamentos del modelo OSI para sistemas de comunicación. Los computadores se interconectan a través de una red de comunicaciones, que puede ser una **red de área amplia** (WAN), cuyo ámbito geográfico es un entorno regional, nacional o internacional, o una **red de área metropolitana** (MAN), que interconecta equipos de una ciudad, o una **red de área local** (LAN), cuyo ámbito es el de un departamento, edificio o campus. En la actualidad también están adquiriendo gran relevancia un tipo de redes inalámbricas de muy pequeño alcance (una habitación o sala, por ejemplo), una red de este tipo se puede denominar **red de área personal**

(PAN). En las secciones 19.1.4 y 19.1.6 describiremos los fundamentos de las redes de área personal, local y de área amplia, respectivamente.

19.1.1 Normalizaciones

Los componentes de un sistema informático (tanto físicos como programas) utilizan **interfases** (Sección 1.2.1); es decir, conjuntos de informaciones (señales o variables) a través de las cuales se pueden interconectar o comunicar unos con otros. Estas informaciones están constituidas de una determinada manera y se pueden utilizar según formas preestablecidas.

Un **protocolo entre dos componentes** *es el conjunto de reglas de procedimiento para conseguir desde uno de ellos, y a través de las interfases respectivas, que el otro realice una función determinada*. Estas reglas definen, para cada componente, la temporización de acciones y respuestas de las diversas informaciones que se pueden intercambiar.

Según las definiciones anteriores las interfases y protocolos especifican la forma de interconectar dos o más componentes de un sistema informático.

Uno de los problemas principales de la interconexión de distintos equipos informáticos (de diversos fabricantes, modelos, etc.) es lograr la utilización de sistemas y procedimientos de intercambio de información comunes. En definitiva, el primer problema es establecer las interfaces y protocolos de interconexión. Las interfaces y protocolos suelen ser fijados por asociaciones internacionales especializadas en estas cuestiones. Concretamente en redes informáticas las dos asociaciones más relevantes son:

- La **Confederación Internacional de Normalización** (**ISO**), en la que están representados diversos países. Dentro de la ISO existen Comités Técnicos, así el nº 97 (ISL/TC 97) se ocupa de normalizaciones en “Computadores y tratamiento de la información” y el nº 16 (ISO/TC 16) de normalizaciones para “Interconexión de sistemas abiertos”.
- El **Comité Consultivo Internacional Telegráfico y Telefónico** (**CITT**), que está formado por los organismos nacionales de los servicios de Correos y Telecomunicaciones. Este Comité (en el que están integrados, entre otras, las siguientes empresas: AT&T de USA, British Telecom del Reino Unido, PTT de Francia y Compañía Telefónica de España) ha elaborado normalizaciones sobre “transmisión de datos a través de red telefónica o télex”, que constituyen la serie V de normalizaciones, y sobre “transmisión de datos sobre redes públicas para la transmisión de datos”, o recomendaciones de la serie X (por ejemplo recomendaciones V24 ó RS-232C, X.25, etc.).

19.1.2 Topologías de redes

Las estaciones (computadores o *hosts*) de una red de computadores se interconectan por medio de una red de comunicaciones (Figura 19.1). Una **red de comunicaciones**, en general, está compuesta de líneas de interconexión y de procesadores de red (Figura 19.2). Las líneas de interconexión son circuitos o canales que interconectan los distintos nodos de la red: computadores (*C-i* en la figura) y procesadores de red (*C-i* en la figura). Los **procesadores de red** son computadores, más o menos complejos, interconectados a dos o más circuitos y que están dedicados a seleccionar el camino (circuito) de salida de la información que le llega por uno de los circuitos de entrada, o a realizar funciones específicamente asociadas al transporte eficiente de la información a través de la red. Los procesadores de red, dependiendo de sus funciones específicas, reciben un nombre u otro, como, por ejemplo: concentrador, conmutador, repetidor, puente, dispositivo de encaminamiento (*router*) o pasarela (Sección 19.2).

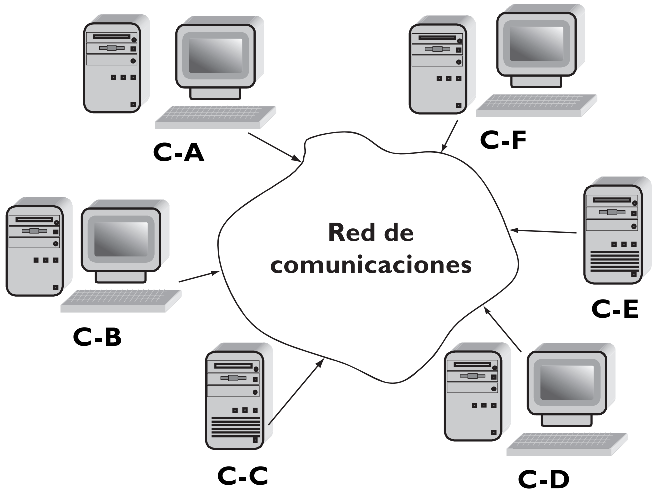


Figura 19.1. *Esquema de una red de computadores.*

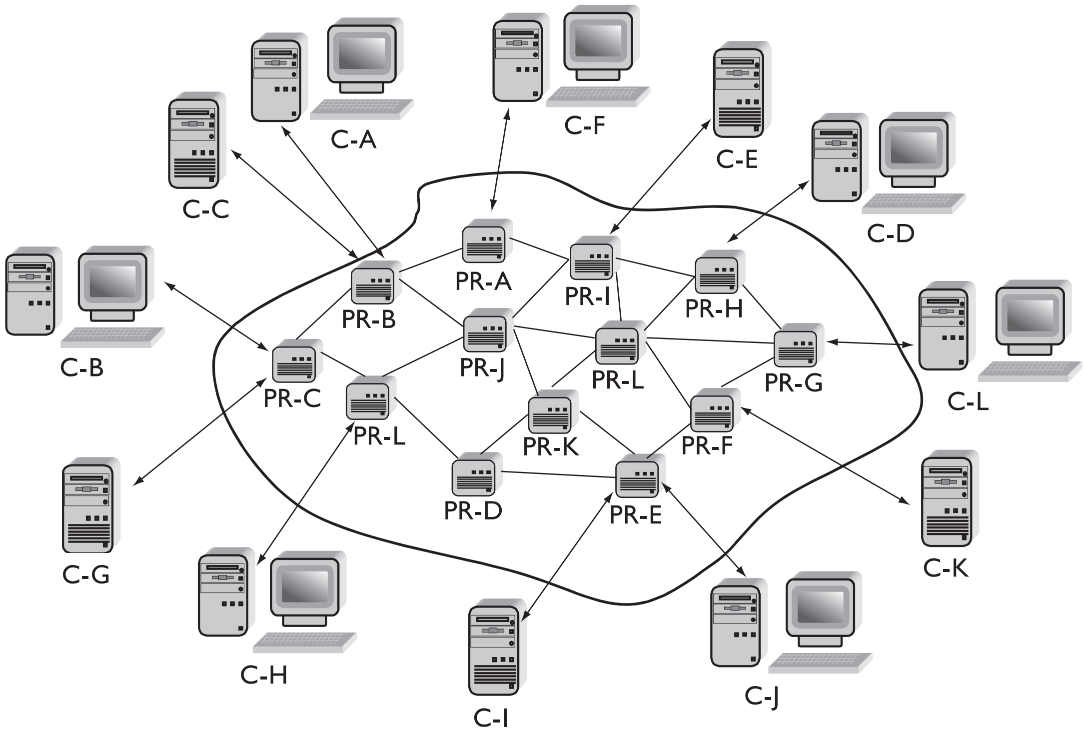


Figura 19.2. *Ejemplo de red de comunicaciones.*

En la Figura 19.3 se representan las *topologías* más usuales de redes de comunicaciones; pudiendo existir en la práctica redes complejas con mezclas de las topologías básicas mostradas en la figura.

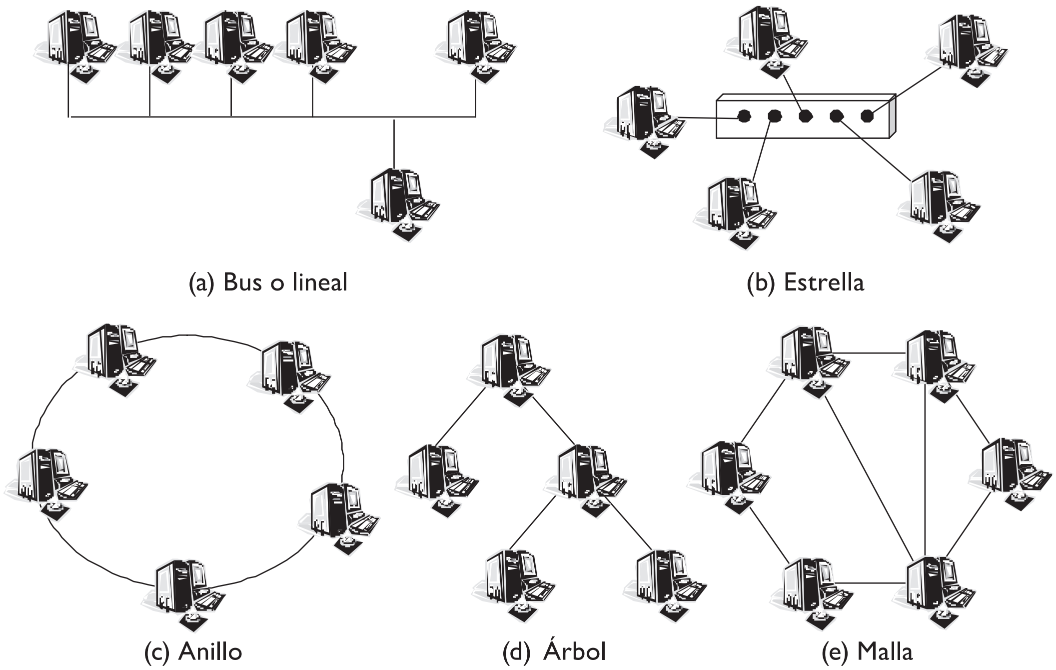


Figura 19.3. *Topologías usuales de redes de comunicación.*

Las *líneas de interconexión*, desde el punto de vista de su utilización, pueden ser de punto a punto o multipunto (Tabla 19.1).

Una línea **punto a punto** o **dedicada** es aquella en la que la transmisión se realiza tan solo entre dos puntos. En este caso la capacidad completa del canal es compartida por los dos nodos que interconecta.

Una **línea multipunto** o **multiconexión** es una línea en la que hay más de dos sistemas compartiéndola. La capacidad del canal es compartida espacial o temporalmente. Los sistemas de multiplexación en frecuencia (radiodifusión) y en longitud de onda (fibra óptica) efectúan una compartición espacial, y los de multiplexación en el tiempo (con asignación de ranuras –*slots*– temporales) efectúan una compartición temporal.

Las **redes punto a punto** suelen utilizar las topologías bipunto, en estrella, en árbol y en anillo. Las **líneas multipunto** son o topologías de bus o topologías en anillo.

Las redes mostradas en la Figura 19.3, salvo la bipunto, son redes de **líneas compartidas** o de **acceso múltiple**, porque en intervalos de tiempo distintos pueden ser utilizadas por diversos nudos.

En un circuito pueden usarse los siguientes *modos de transmisión* (Tabla 19.1):

a) **Simplex** (Figura 19.4a). Se transmite en una sola dirección. Es el caso de la conexión de datos entre un computador y una impresora; los hilos de datos están dedicados a la transmisión en un solo sentido (computador a impresora).

b) **Semidúplex** (Figura 19.4b). El mismo canal o línea se utiliza para transmitir en los dos sentidos, pero no simultáneamente. Unas veces se transmite en un sentido y otras en el contrario, necesitándose conmutadores en ambos extremos para utilizar la línea en una u otra dirección.

c) **Dúplex** (Figura 19.4c). Se transmite simultáneamente en ambos sentidos. Es el caso de la conexión de datos de un terminal interactivo con un computador, en la que hay dos pares de hilos para transmisión de datos: uno dedicado a recepción y otro a transmisión. También puede usarse el canal con multiplexación en frecuencia, modulando las señales de recepción y transmisión de distintas frecuencias de portadora.

Tabla 19.1. *Configuraciones de las redes de comunicaciones.*

Área geográfica cubierta	– Red de área personal (PAN) – Red de área local (LAN) – Red de área metropolitana (MAN) – Red de área amplia (WAN)
Uso de las líneas	– Línea punto a punto o dedicada – Línea multipunto o multiconexión
Topologías	– Bipunto – Bus o lineal – Estrella – Árbol – Anillo – Malla – Malla totalmente conectada o de barras cruzadas
Modos de transmisión en un canal	– Simple – Semi-dúplex – Dúplex

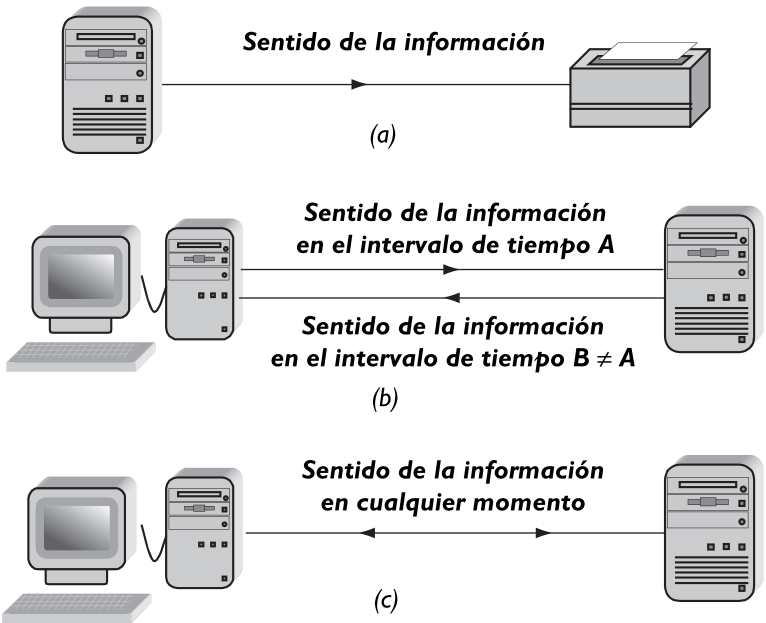


Figura 19.4. *Enlaces (a) simple; (b) semidúplex; y (c) dúplex.*

19.1.3 Modelo OSI

El proceso de transmisión de datos es sumamente complejo, interviniendo componentes software y hardware, así como medios, muy diversos. Debido a ello los procedimientos que tienen lugar se suelen estructurar en capas o niveles. Esta forma de organizar el proceso completo de transmisión se suele realizar en la vida cotidiana en casi todos los procesos de comunicaciones. Pensemos, por ejemplo, en un departamento, *DA,* de una empresa, *EA*, que envía una carta con documentos al departamento *DB* de la empresa *EB*. Muy posiblemente el proceso será el siguiente:

- El empleado que envía la carta, adecuadamente cerrada y con la dirección de destino claramente especificada en su exterior, la deposita en la secretaría de su departamento (*DA*), inhibiéndose desde ese momento de la responsabilidad del envío.
- Un ordenanza del departamento, con otro conjunto de cartas de éste, las lleva a la conserjería de la empresa donde se centraliza la recogida de todas las cartas de los distintos departamentos.
- Una persona de conserjería lleva toda la correspondencia a una estafeta de correos próxima.
- En la estafeta de correos recopilan las cartas del distrito postal correspondiente y las clasifican e introducen en distintas sacas según el destino.
- Las sacas son llevadas por los medios de transporte establecidos (camión, tren, avión, etc. o mezcla de estos medios) a la ciudad de destino.
- Etc.

En la ciudad de destino se efectúa el proceso inverso: se abren las sacas, se clasifican las cartas por calles, un cartero las lleva a la conserjería de la empresa *EB*, un responsable del departamento *DB* recoge las cartas, etc.

El proceso anterior se puede descomponer conceptualmente en capas, de acuerdo con una jerarquía. La capa de mayor nivel sería la correspondiente a los usuarios del transporte de correos (capa de aplicación), que utiliza los servicios de la capa inmediata inferior (secretaria del departamento), quien, a su vez, utiliza los servicios ofrecidos por la capa inferior (conserjería de la empresa), y así sucesivamente. Podemos decir que entre el empleado que envía la carta desde el departamento *DA* se establece una **comunicación virtual** con el empleado a quien va destinada la carta en el departamento *DB*, ya que la **comunicación real** o física es efectuada por la empresa de correos que transporta las sacas de una ciudad a otra. Por otra parte el objetivo de cada capa se limita a recoger la *unidad de transporte* que le proporciona la capa superior, efectuar con ella alguna manipulación sencilla, y entregarla a la capa inmediata inferior. En otras palabras cada capa está especializada en una función relativamente sencilla.

De la misma forma podemos considerar la interconexión entre dos computadores *A* y *B* estructurada en capas. La comunicación se lleva a cabo utilizando cada capa $i + 1$ los servicios proporcionados por la capa inmediata inferior i. La comunicación *real* se efectúa únicamente a través de la capa *1*, que representa el substrato físico (medio de comunicación). La comunicación directa dentro del mismo nivel entre el sistema *A* y el *B* es una comunicación virtual. Con este modelo el diseño de un sistema de transmisión de datos entre computadores se simplifica notablemente, ya que cada capa, en el emisor, es únicamente responsable de tomar la información de la capa inmediata superior y proporcionarla, adecuadamente transformada, a la inmediata inferior: los problemas de diseño y funcionamiento quedan aislados entre capas.

En el receptor el proceso es el inverso: una capa *i* toma la información de la capa inmediatamente inferior *i – 1* y la proporciona a la capa inmediatamente superior *i + 1*, después de las comprobaciones y transformaciones oportunas.

El **Modelo de Referencia OSI** (*Open Systems Interconnection*), propuesto por ISO, utiliza la sistemática anteriormente descrita, en la Figura 19.5 puede verse un esquema simplificado del mismo. En los extremos figuran los sistemas entre los que se requiere efectuar la conexión (Sistemas *A* y *B*). En el centro figuran otros elementos que intervienen sólo con objeto de efectuar la transmisión (fundamentalmente procesadores de comunicaciones).

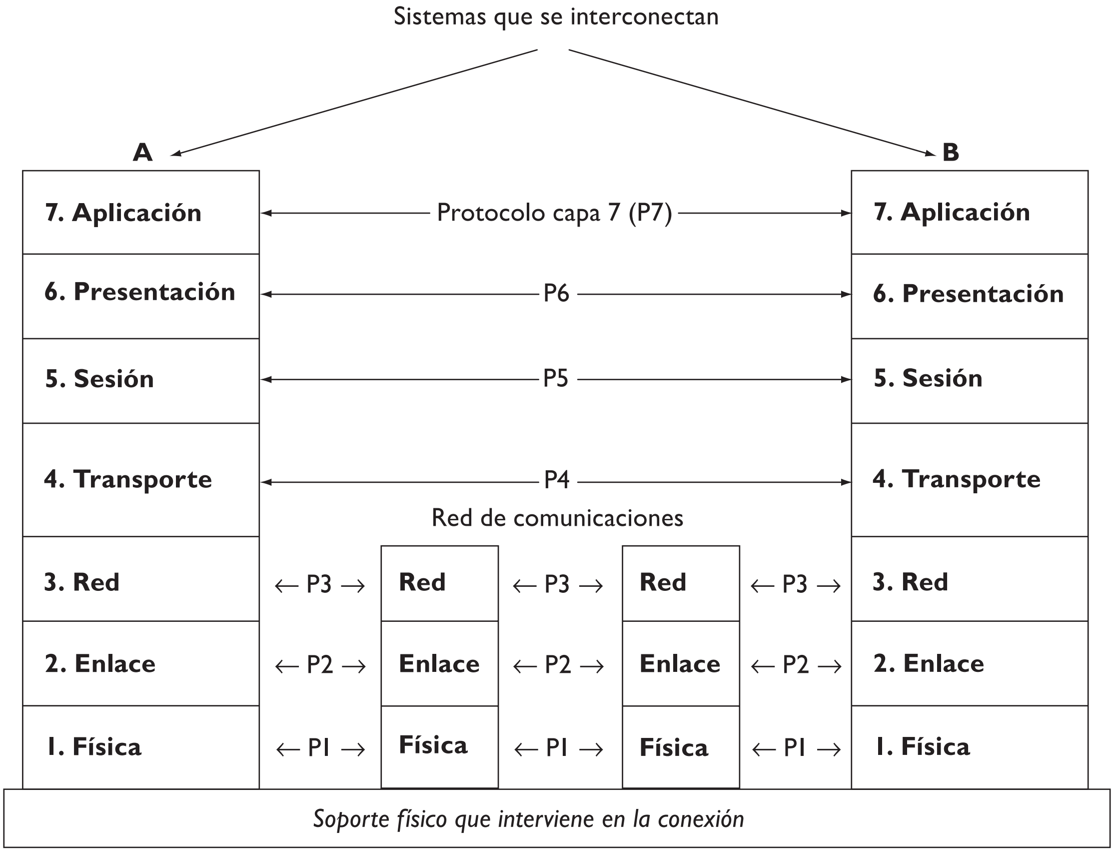


Figura 19.5. *Esquema simplificado del modelo de referencia ISO, para interconexión de sistemas abiertos.*

En el modelo ISO se consideran siete niveles conceptuales. En cada uno de ellos se procesan unidades de información denominadas **PDU** (**Unidad de datos de protocolo**). En el computador emisor las PDU se transmiten del nivel superior al inferior, y en cada uno de ellos se añade información de control (**cabeceras**, *AH, PH, SH, TH* y *DH*, en la Figura 19.6, o terminales, *DT*). En el computador receptor la información se procesa desde el nivel inferior (que es por el que físicamente se transmite la información) hasta el superior, comprobándose y eliminándose en cada nivel las cabeceras o terminales de cada PDU correspondientes a dicho nivel. A continuación se describen brevemente las funciones de cada nivel (Figuras 19.5 y 19.6).

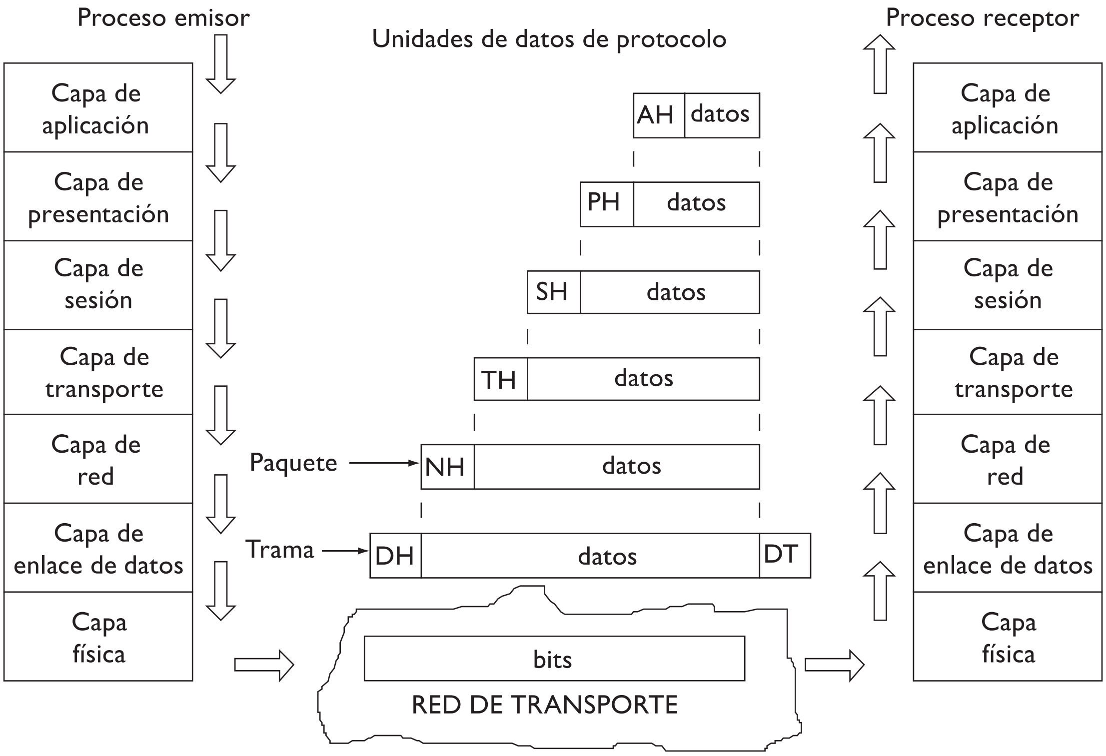


Figura 19.6. *Unidades de información utilizadas en cada capa ISO.*

Capa 7. Capa de aplicación

Dos computadores, *A* y *B*, se intercomunican a través de procesos, *PA* y *PB*, correspondientes a unas determinadas aplicaciones. El intercambio de información entre los dos procesos se efectúa por medio de algún protocolo de la capa de aplicación. Así, por ejemplo, si se utiliza una aplicación de correo electrónico (Eudora o Outlook, por ejemplo) en la máquina *A*, ésta necesitará en algún momento intercambiar información con otro computador *B*, para lo cual utilizará, por ejemplo, el protocolo SMPT de la capa de aplicación (Sección 19.2.2). Las aplicaciones de los usuarios que se intercomunican a través de una red con otras necesitan servicios (protocolos) para realizar tareas tales como acceso remoto, transferencia de archivos y correo electrónico.

EJEMPLO 19.1

HTTP (Protocolo para transferencia de hipertextos) es un protocolo de la capa de aplicación que permite a un navegador de web, proporcionando un nombre, solicitar a un servidor web una determinada página.

Capa 6. Capa de presentación

El nivel de presentación trata de homogeneizar las formas (formatos) de representación de los datos entre equipos de la red. Como se vio en el Capítulo 4, existen muchas formas de representar

tanto los caracteres como los datos numéricos, siendo éstas dependientes del computador. Para homogeneizar la representación de datos, la capa de presentación interpreta las estructuras de las informaciones intercambiadas por los procesos de la aplicación y las transformarlas convenientemente. Además puede realizar transformaciones para conseguir una mayor eficiencia de la red (compresión de texto y cifrado de seguridad). Los programas correspondientes a este nivel suelen incluirse en el propio sistema operativo, o, alternativamente, constituir una biblioteca de programas específica del nivel. En otras palabras, esta capa proporciona a los procesos de aplicación independencia respecto a las diferencias en la representación de los datos.

EJEMPLO 19.2

*El **XDR** (**EX**ternal **D**ata **R**epresentation) es un formato desarrollado por Sun para la descripción y representación de datos, basada en los tipos de datos del lenguaje C, que se ha ideado para el intercambio de información entre procesos remotos. Es, por tanto, un elemento de la Capa 6. Cuando se envía un mensaje de un proceso en un computador A a otro de un computador B, en la capa 6 del computador A la información se convierte en datos en formato XDR, y en él se transmite la información. Cuando el mensaje llega a la capa 6 del computador B, se realiza un proceso inverso: se convierte la información del formato XDR al propio del computador B.*

Capa 5. Capa de sesión

Cuando se realiza una transferencia entre dos sistemas o usuarios se establece una **sesión de comunicaciones** entre ambos. La capa de sesión es responsable de funciones tales como:

- Actuar de interfaz entre el usuario y la red, gestionando el establecimiento de la conexión entre procesos remotos.
- Establecer un dialogo entre dos equipos remotos para controlar la forma en que se intercambian los datos.
- Identificar los usuarios de procesos remotos.
- Cuando se rompe anómalamente una conexión, en la capa de transporte, o en capas inferiores, la capa de sesión puede encargarse de restablecerla de forma transparente al usuario.

En definitiva, aumenta la fiabilidad de la comunicación obtenible por las capas inferiores, proporcionando el control de la comunicación entre las aplicaciones al establecer, gestionar y cerrar las sesiones (conexiones) entre las aplicaciones que se comunican.

Capa 4. Capa de transporte

Este nivel se encarga del transporte de la información, desde la fuente al destino, a través de la red. Los accesos a la capa de transporte se efectúan a través de puertos (usualmente ***sockets***, definidos en el UNIX de Berkeley), cuyos nombres se dan con un criterio común a todos los usuarios. El objetivo fundamental es realizar un servicio de transporte eficiente entre procesos (usuarios) finales. Para ello toma los **mensajes** del nivel sesión, los distribuye en pequeñas unidades, denominadas **segmentos** (Figura 19.6, PDU, TH), y los pasa a la capa red. Los protocolos de la capa de transporte se aseguran de que todas las unidades lleguen correctamente, para lo cual realizan detección y corrección de errores, además de controlar el flujo y la secuenciación. De esta forma también se consigue aislar la capa de sesión de la capa de red e inferiores, haciendo aquella capa independiente del hardware específico. Este nivel también se encarga

de optimizar el transporte; realizando, por ejemplo, multiplexaciones de varios mensajes en un segmento para abaratar los costos; estas multiplexaciones son transparentes a los niveles superiores. También, para aumentar la velocidad de transmisión, puede usar varias conexiones de la capa de red para la misma conexión de la capa de transporte.

Capa 3. Capa de red

La capa de red se encarga de:

a) **Fragmentar** las informaciones (segmentos) que se transmiten entre dos equipos de datos en unidades denominadas **paquetes** (PDU, NH, Figura 19.6). En la estación destino se efectúa el proceso inverso: los paquetes se ensamblan en segmentos.
b) Realizar el **encaminamiento** de los paquetes. En efecto, los paquetes para ir de un computador a otro (de C-A a C-I, por ejemplo, en la Figura 19.2) pueden seguir distintas rutas (PR-B, PR-J, PR-K, PR-E o PR-B, PR-C, PR-L, PR-D, PR-E, por ejemplo), el nivel que nos ocupa se encarga de utilizar algoritmos eficientes para la elección de la ruta más adecuada en cada momento, y, de acuerdo con ella, reexpedir los paquetes en cada uno de los nudos de la red que deba atravesar.
c) Prevenir la producción de bloqueos así como la congestión en los nudos de la red de transporte que pudiesen producirse en horas punta por la llegada de oleadas de paquetes.

Cada nodo intermedio de la red (que puede ser un procesador de comunicaciones o un computador convencional) debe asumir la responsabilidad de la capa de red y sus inferiores.

Capa 2. Capa de enlace de datos

Esta capa descompone los mensajes que recibe del nivel superior en **tramas** o bloques de información, en las que añade una cabecera (*DH*) e información redundante para control de errores (*DT*, Figura 19.6). La cabecera suele contener información tal como direcciones de origen y destino, información sobre la ruta que va a seguir la trama, etc. Estas operaciones se realizan en los equipos terminales y en los procesadores de comunicaciones. También se encarga de transmitir sin error las tramas entre cada enlace que conecte directamente dos puntos físicos (nodos) adyacentes de la red, y desconectar el enlace de datos sin pérdidas de información. Si el medio de comunicación es de difusión (radio, Ethernet, etc.), esta capa se encarga también del control de acceso al medio.

Nivel 1. Capa física

En esta capa es dónde se especifican los parámetros mecánicos (grosor de los cables, tipo de conectores, etc.), eléctricos (temporización de las señales, niveles de tensión, por ejemplo), etc., de las conexiones físicas. Las unidades de información que considera son **bits**, y trata de lo concerniente a la transmisión de cadenas de bits en el canal de comunicación (pares trenzados de cobre, cable coaxial, radio, infrarrojos, o fibra óptica): si en el emisor se envía un 1, al receptor debe llegarle un 1.

Hay que indicar que el Modelo ISO es un modelo conceptual que no define ni especifica interfaces y protocolos, únicamente establece criterios generales sobre cómo concebir las redes de comunicaciones de datos.

19.1.4 Redes de área personal

Una red que cubre un área muy pequeña, tal como una habitación o una sala, puede denominarse red **de área personal** o **PAN** (*Personal Area Network*). Estas redes, en general, tratan de evitar la utilización de cables para interconectar los periféricos a un computador o distintos sistemas

de electrónica de consumo entre sí. De esta forma, por ejemplo, se evita a los usuarios tener que conectar a la unidad central los distintos cables procedentes del monitor, ratón, teclado, impresora, escáner, etc. En definitiva se trata de que cuando una persona adquiera un computador o un periférico nuevo los conecte a la red eléctrica y funcionen sin más, sin necesidad de asistencia técnica.

Estas redes suelen utilizar radio de corto alcance o infrarrojos. En la sección siguiente, dentro del apartado dedicado a redes LAN inalámbricas (Sección 19.1.5.3), se analizarán las redes *Bluetooth,* de infrarrojos y de microondas de banda estrecha, que son las tecnologías que se utilizan para redes de área personal.

19.1.5 Redes de área local

Las características fundamentales de una red de área local son: longitud de línea de unos metros (una habitación) a unos pocos kilómetros (campus universitario, por ejemplo), velocidad de transmisión de datos usuales del orden de 10 a 100 Mbps, y los equipos y red suelen atender a una única empresa o entidad (son redes privadas).

La red local puede tener nudos de conexión a otras redes locales o a redes públicas de área amplia (Sección 19.1.6). Desde un terminal de datos de la red se pueden utilizar todos los recursos de la red local y de las redes conectadas a ésta.

Las redes de área local se suelen clasificar en dos tipos: redes que acceden a la línea por escucha y redes en forma de anillo. Las topologías que se suelen utilizar (Figura 19.3) son: la de bus, la de árbol, la de anillo y de estrella. En esta última la red actúa en modo de conmutación de circuito y hay un nudo principal que establece en realidad un enlace del tipo bus lineal. También una red en forma de anillo se puede considerar como un caso particular de topología en estrella.

En las redes locales, para compartir la línea o líneas entre distintos equipos, se suelen utilizar **métodos de escucha**, que son métodos distintos a los utilizados por las redes públicas. Los métodos de escucha pueden clasificarse en **métodos de acceso múltiple sensible a portadora o métodos CSMA**, **método Ethernet** y **métodos de anillo**.

Los métodos CSMA presuponen que los terminales emiten señales moduladas. Un equipo que desea emitir, primero comprueba ("escucha") si hay portadora en la línea, si no es así es que la línea está libre y emite sin más. Cuando dos o más equipos están esperando a que la línea esté libre y en el momento en que queda disponible tratan simultáneamente de emitir, se dice que se ha producido una **colisión**, que se resuelve de distintas formas, dependiendo del tipo de red.

19.1.5.1 Red Ethernet (IEEE802.3)

En una red Ethernet, cuando una emisora transmite, a la vez lee la información que hay en la línea y si no coincide lo transmitido con lo leído, evidentemente es que se ha producido una colisión, destruyéndose los paquetes en conflicto. Esto proporciona una ventaja sobre otros procedimientos de acceso a línea utilizados, en los que es necesario esperar a que llegue un paquete de reconocimiento para saber si se ha producido o no una colisión. En la red Ethernet se detecta inmediatamente la colisión, interrumpiéndose en ese momento la emisión de los paquetes destruidos. Una vez que se produce, se reintenta emitir después de transcurrido un tiempo aleatorio. Este procedimiento de acceso a la red de denomina CSMA sensible a portadora (**CSMA/CD**, siglas que corresponden a *Carrier Sense Multiple Access with Collision Detection*, en español, **Acceso Múltiple con Escucha de Portadora y Detección de Colisiones**).

La red Ethernet original (desarrollada por Xerox Corp.) es de topología de bus. Utiliza emisiones en banda base (es decir, en realidad no hay portadora) con código de línea Manchester a 10 Mbps; la interconexión se hace con cable coaxial grueso (0,4 pulgadas, red de modalidad 10Base5) o fino (0,25 pulgadas, modalidad 10Base2 o **Cheapernet**).

Los paquetes de información contienen un campo de datos de 368 a 16.000 bits, y otros campos de sincronismo (64 bits), destino (48 bits), origen (48 bits), tipo de protocolo utilizado (16 bits) y código CRC de detección de errores (32 bits).

Existen dos topologías para Ethernet, que se usan según la versión (Tabla 19.2):

- *Topología de bus.*

  Es la usada por las primeras redes Ethernet. Estas redes se forman con segmentos unidos por repetidores. Todos los circuitos son de cable coaxial (Figura 19.7a), existiendo una separación mínima entre tomas de 2,5 metros y la distancia entre la toma del bus común y el computador puede llegar a 50 metros.
- *Topología en estrella.*

  Es la más utilizada en la actualidad. Todas las estaciones de un segmento se conectan por medio de pares trenzados (10Base-T, 100Base-T, 1000Base-T y 10GBase-T, sin apantallar –UTP– o apantallados –STP–, dependiendo de la versión, y de hasta una longitud de 100 metros) o fibras ópticas a un panel de conexiones o concentrador común (véase Figura 19.7b y Tabla 18.3).

La tecnología Ethernet más avanzada es la **Gigabit Ethernet** (**GigE**) que fue definida (IEEE 802.3z) para usar con canales de fibra óptica y compartiendo el medio con protocolo CSMA/CD. Posteriormente se definió un estándar (802.3ab, 1000Base-T) que permite usar la red con canales de hilo de cobre pudiendo así transformar fácilmente una Fast Ethernet en una Gigabit Ethernet.

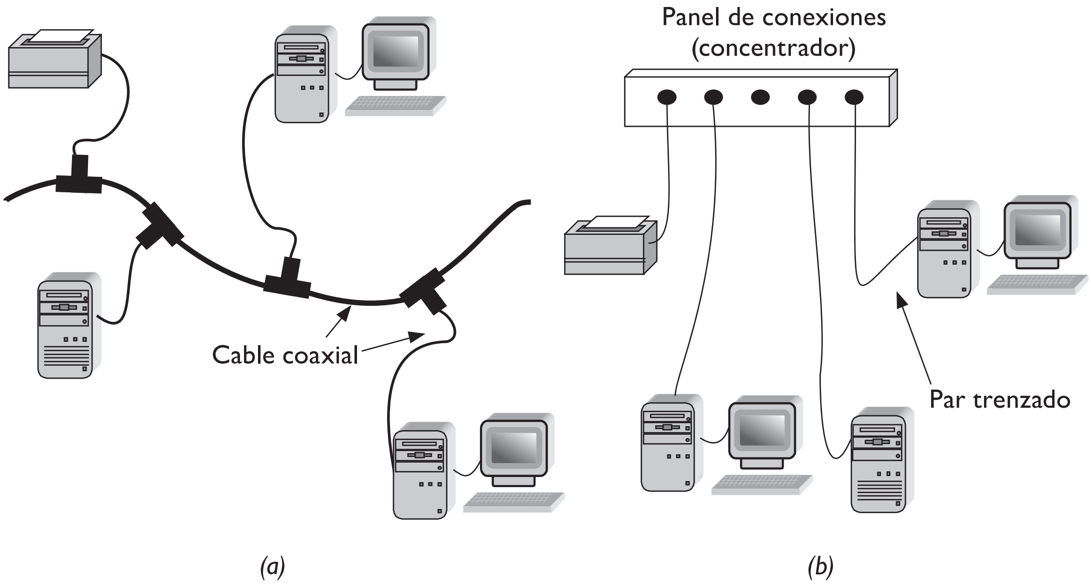


Figura 19.7. *Red Ethernet (a) con topología de bus (cable coaxial); (b) con topología en estrella (pares trenzados).*

En la Tabla 19.2 se enumeran las principales versiones de redes Ethernet.

Tabla 19.2. *Distintas versiones de redes Ethernet (Estándar IEEE 802.3).*

Banda base digital	– Ethernet (Manchester)	• 10Base5 (10Mbps, topología de bus, coaxial grueso) • 10Base2 Cheapnet (10Mbps, topología de bus, coaxial fino) • 10Base-T (10Mbps, topología en estrella) • 1Base-5 (1Mbps, topología en estrella, pueden conectarse de forma encadenada hasta 10 equipos en una única toma del concentrador)
	– Fast Ethernet (100Mbps, topología en estrella)	• 100BaseT4 (4 pares no apantallados, 2 para recepción y 2 para transmisión) • 100BaseTX (2 pares apantallados o sin apantallar) • 100BaseFX (2 fibras ópticas)
	– Gigabit Ethernet (1Gbps, topología en estrella)	• 1000BaseSX (fibra óptica, longitud de 300 metros) • 1000BaseLX (fibra óptica, longitud de 550 metros –multimodo–, o 5 Km –monomodo) • 1000BaseCX (cable blindado, 25 metros, cluster de computadores) • 1000BaseTX (4 pares trenzados sin apantallar, longitud 100 metros).
Banda ancha analógica	– 10GEthernet – 10Broad36 (10 Mbps, codificación PSK)	

19.1.5.2 Redes en anillo

En las redes en anillo (Figura 19.8) cada computador está unido con dos conexiones punto a punto a dos estaciones adyacentes. La unión entre dos estaciones vecinas se denomina segmento.

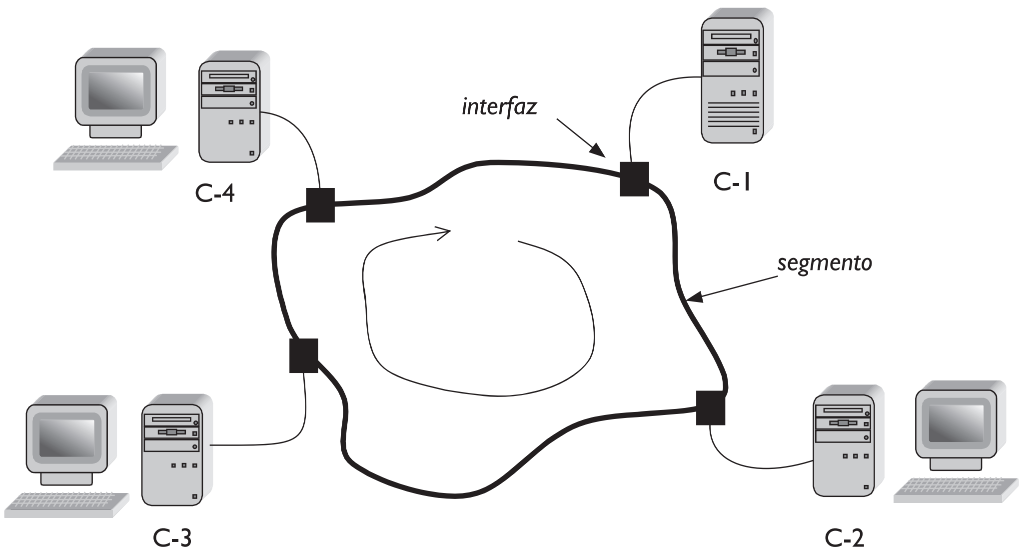


Figura 19.8. *Esquema simplificado de una red en anillo.*

Usualmente los mensajes se transmiten en un único sentido. Supóngase (Figura 19.8) que el computador *C-1* envía un mensaje a *C-3*. Primero lo detecta *C-2*, lo reexpide, después llega a *C-3*, que lo capta (por ser un mensaje dirigido a ella) y lo reexpide (dejándolo como está o cambiando alguno de sus campos). *C-4* lo capta (procedente de *C-3*) y lo reexpide hacia *C-1*. *C-1*, es decir, la fuente del mensaje, al recibirlo, comprueba si coincide con lo que ha enviado o detecta los cambios introducidos por *C-3*, y lo destruye. De esta forma el emisor acusa recibo de su propio mensaje.

Hay diversas formas de gestionar el tráfico de paquetes en redes en anillo, únicamente se citan a continuación las cuatro técnicas consideradas más clásicas:

Técnica del testigo o anillo de Newhall *(token rings)* (IEEE 802.5 ó ISO 8802.5)
En un momento dado sólo puede circular un mensaje por el anillo. Cuando no hay datos a transmitir por el anillo circula un **testigo**, que es una cadena de bits determinada; por ejemplo, 1111 1111. Para no confundir esta cadena con bits correspondientes a datos, en estos últimos se añaden bits ficticios que son posteriormente eliminados en el receptor, de forma que nunca se pueda presentar una secuencia de datos que coincida con la secuencia del testigo. Cuando una estación desea emitir un mensaje, espera a recibir el testigo, y antes de reexpedirlo lo cambia por una cadena de *conexión* (por ejemplo 1111 1110). Inmediatamente después inserta el paquete a transmitir. Al llegar a su destino, el mensaje es captado, pero sigue circulando hasta el emisor (se completa el anillo). El emisor restituye el testigo y elimina el mensaje. En estas redes clásicamente se obtienen velocidades de 4 ó 16 Mbps.

Técnica de testigo con contención
En la técnica anterior se producen problemas si el testigo, por error en la línea, se pierde, y, aunque cualquier estación puede regenerarlo, puede darse el caso de que dos lo regeneren no simultáneamente con el consiguiente conflicto. La técnica de contención no utiliza testigo cuando la línea está libre. Si una estación desea emitir lo hace cuando observa que la línea está libre. Al final del paquete incluye un testigo. Si el testigo da la vuelta completa (por no haberlo cogido alguna de las otras estaciones), la estación que lo generó lo elimina.

Técnica del anillo con ranuras temporales o anillo de Pierce
Con esta técnica en el anillo puede llegar a haber simultáneamente tantos mensajes como segmentos. Los paquetes son todos de igual longitud, y constantemente por el anillo están circulando "contenedores" o "ranuras" (*slots*) para paquetes, que pueden estar vacíos o llenos. En su cabecera va un bit especificando esta última circunstancia. Cuando una estación desea emitir, espera hasta que llegue un contenedor vacío, cambia el bit de control de libre a ocupado e inserta su paquete. Cuando vuelve a recibir su paquete, el emisor, si no desea seguir emitiendo paquetes, cambia el bit de ocupado a libre (una vez verificado que ha dado la vuelta correctamente). De esta forma el contenedor queda libre para volver a ser ocupado por la misma o por otra emisora.

Redes FDDI *(Interfaz de distribución de datos con fibra)*
Fueron proyectadas para fibra óptica, pero pueden usarse en la actualidad con hilos de cobre. Se consiguen velocidades de hasta 100 Mbps. Las estaciones pueden emitir dos tipos de tramas: asíncronas (tramas-A) y síncronas (tramas-S). Las estaciones pueden enviar tramas siguiendo un turno rotatorio (primero el computador *C-1*, luego el *C-2*, *C-3*, *C-4*, *C-1*, etc.) y se establece el tiempo medio, *TTRT*, del intervalo que transcurre desde que se le da un turno a un computador hasta que se le vuelve a dar de nuevo. Además se asigna un tiempo, *SA*, para que cada estación envíe tramas-S (que no tienen por qué ser iguales para todas ellas). Cuando a una estación le llega su turno (el testigo) puede enviar durante SA segundos tramas síncronas

(si hubiese pendientes de enviar), y el resto de tiempo hasta *TTRT* las tramas asíncronas que tuviese a la espera de enviar. Con este sistema se hace un autocontrol de la red, obteniéndose un uso equilibrado del bus de la red.

Las redes FDDI contienen dos anillos que generalmente funcionan uno en un sentido y el otro en sentido contrario. Si uno de los anillos falla, un sistema puede seguir comunicando con otro.

La Tabla 19.3 muestra un resumen de las redes LAN descritas.

Tabla 19.3. *Características de varias LAN.*

	Ethernet	Anillo con testigo	FDDI
Método de acceso	CSMA/CD	Paso de testigo	Paso de testigo
Velocidad de transferencia	1, 10, 100 Mbps	10 a 16 Mbps	4, 16, 100 Mbps
Control de errores	No	Sí	Sí
Medio de transmisión	Cable coaxial, par trenzado y fibra	Par trenzado o fibra	Fibra óptica y par trenzado

19.1.5.3 Redes LAN inalámbricas (Wi-LAN)

En general, una red inalámbrica está formada por un conjunto de **módulos de control**, cada uno de los cuales esta conectado a una red cableada troncal (tal como una Ethernet) y cubre un determinado espacio denominado **celda**. Cada estación o equipo terminal de la red debe quedar dentro de una celda. Si el área a cubrir es relativamente pequeña (una habitación o red de área personal, por ejemplo) la red puede disponer de tan sólo un módulo de control.

En la actualidad la mayor parte de las redes inalámbricas locales siguen el conjunto de normalizaciones IEEE 802.11, y cuando un producto sigue correctamente las mismas puede obtener un certificado de **red Wi-Fi** (*Wireless Fidelity*). Hoy en día las redes Wi-Fi también se utilizan para acceder a Internet.

Como veremos en esta misma sección, una red Wi-Fi (es decir, una red que sigue alguna de las normalizaciones IEEE 802.11), tiene un alcance reducido, de unos 250 metros en espacio abierto. En el 2001 se constituyó una agrupación para certificar redes que siguen la normalización IEEE 802.16 para redes de banda ancha, en cuyo caso permiten utilizar el logotipo **WiMAX** (*World Interoperability for Microwave Access*). Una **red WiMAX** puede considerarse como una Wi-Fi optimizada ya que con ella se obtienen alcances de hasta 48 kilómetros de radio con unas tasas de datos de hasta 70 Mbps, características sumamente atractivas para, por ejemplo, empresas proveedoras de servicios Internet. En realidad una WiMAX, dado su alcance, es una **red de área metropolitana** (**MAN**), y está proyectada la incorporación próxima de esta tecnología a teléfonos móviles, computadores portátiles y sistemas PDA. Con ello se facilitará la conexión a Internet de los sistemas citados, y se utilizará el estándar WiMAX Móvil (802.16e) para terminales en movimiento.

A continuación se describen brevemente distinto tipos de redes LAN: infrarrojos, espectro expandido, microondas de banda estrecha, Bluetooth y ZigBee.

Infrarrojos (IR, Infrared)

Como su nombre indica, estas redes utilizan la banda del espectro electromagnético de los infrarrojos, que está en el límite del espectro visible, y se usan mucho en los hogares como redes de

área personal para el control remoto de muchos dispositivos (mando a distancia de televisores, aparatos de audio, etc.). En la actualidad se están utilizando también para configurar redes LAN inalámbricas. Dada la alta frecuencia de los infrarrojos (300 GHz a 360 THz) la información transmitida por medio de ellos puede ir a velocidades extraordinariamente elevadas. Por otra parte, como ocurre con la luz, la radiación infrarroja se refleja con gran facilidad y no atraviesa objetos opacos (como puede ser una pared). Estas propiedades tienen el inconveniente de que los infrarrojos sólo son útiles para distancias muy cortas (de 15 a 60 metros), y puede haber radiación de fondo de la luz solar o artificial que interfiera con el haz que transmite la información. Por el contrario, la transmisión por medio de infrarrojos presenta las siguientes ventajas: fiabilidad frente a escuchas indeseadas y a interferencias radioeléctricas, y al reflejarse fácilmente puede ser captada con cierta facilidad por los receptores que estén, por ejemplo, en una habitación. También, las instalaciones son baratas y relativamente sencillas, pudiendo utilizarse en conexiones punto a punto; de esta forma se puede diseñar, por ejemplo, una LAN de anillo con paso de testigo dentro de una sala o habitación. También se puede construir en una sala una red omnidireccional ubicando, por ejemplo en el techo, una unidad base que actúe como repetidor que difunda la señal en todas las direcciones. Las unidades IR en los equipos de los usuarios apuntarían a la unidad de control central del techo para, a través de ella, comunicarse con el resto de equipos de la red. Si no fuese necesaria la unidad de control por no desear comunicación con otras redes, bastaría con que todos los transmisores-receptores IR apuntasen a una zona concreta del techo y así por reflexión se podrían intercomunicar con el resto de las unidades (**configuración IR de difusión**).

Con infrarrojos difusos se logran tasas de datos de 1 a 4 Mbps y alcances de 15 a 60 metros, mientras que con infrarrojos de haz directo las velocidades están comprendidas entre 1 y 10 Mbps y el alcance puede llegar a los 25 metros. La modulación que se utiliza es ASK (véase Sección 18.2.3.2) y el método de acceso que suele utilizarse es CSMA o anillo con paso de testigo. El estándar IEEE 802.11 establece velocidades de 1 y 2 Mbps funcionando en el rango de 3,33 a 375 THz.

Espectro expandido

La mayoría de las redes LAN de espectro expandido actúan en bandas de frecuencia asignadas a industria, ciencia y medicina (ISM), por lo que no necesitan licencia para su utilización.

Por lo general, estas redes se estructuran en diversas celdas, utilizando celdas adyacentes con frecuencias distintas, aunque dentro de la misma banda. De esta forma se evitan interferencias.

Pueden usarse dos configuraciones. En la primera de ellas las distintas estaciones de la red tienen antenas omnidireccionales comunicándose entre ellas de igual a igual (*peer to peer*); la situación es análoga a la transmisión de varias estaciones a través de un bus único (acceso CSMA). En la segunda configuración un módulo de control actúa como repetidor o concentrador multipunto y todas las estaciones en la celda intercambian la información (reciben y transmiten) con otras estaciones de la red a través del repetidor, que, a su vez, puede estar conectado a una red cableada.

La denominación de **espectro expandido** (por salto de frecuencia, o **FHSS**) hace referencia a que cada celda hace uso de varios canales de frecuencia, saltando la señal de un canal a otro de acuerdo con una secuencia pseudoaleatoria conocida tanto por la unidad de control como por las estaciones de su celda. El emisor y receptor van cambiando sincronizadamente las frecuencias de tal forma que se produce correctamente la recepción. Con esta técnica (que utiliza un espectro mayor del estrictamente necesario) se reducen posibles interferencias entre celdas adyacentes o con otras fuentes de radiación y, además, aumenta la invulnerabilidad y privacidad de la red.

Las tasas de datos obtenibles con estas redes están comprendidas entre 1 y 50 Mbps, con alcances de 30 a 250 metros y bandas de frecuencias de 902 a 928 MHz, 2,4 a 2,4835 GHz y 5,725 a 5,85 GHz. El estándar IEEE 802.11 establece para esta forma de transmisión un funcionamiento en la banda de los 4 GHz, con tasas de datos de 1 y 2 Mbps.

EJEMPLO 19.3

En la mayor parte de Europa, en las redes LAN de espectro expandido del tipo FHSS, como mínimo se generan 2,5 saltos por segundo y la distancia mínima de cada salto de frecuencia es de 6 MHz.

Redes LAN de microondas de banda estrecha

Lo usual en estas redes es utilizar la zona ISM del espectro que, como se ha comentado anteriormente, no necesita licencia de uso. Pueden llegar a una tasa de datos de unos 10 Mbps, en la banda de los 5,8 GHz, alcanzando unos 50 metros en espacios cerrados y 100 metros en zonas abiertas.

El estándar 802.11b ha adquirido una gran popularidad, se asocia con la denominación Wi-Fi, y establece tasas de datos entre 5,5 y 11 Mbps. Utiliza como forma de acceso al medio CSMA/CA (*Carrier Sense Multiple Access/Collision Avoidance*), en la banda de 2,4 GHz. Esta tecnología de transmisión trata de *evitar* las colisiones producidas en el medio, en lugar de *detectarlas*, como hace la CSMA/CD utilizada en las redes Ethernet. Debido a la sobrecarga producida por los bits de control adicionales en las tramas de transmisión de datos del usuario, se consiguen tasas de datos más bajas que las teóricas: 5,9 Mbps con TCP y 7,1 Mbps con UDP.

En el año 2003 se aprobó una mejora del estándar 802.11b, denominada 802.11g, con la que la tasa de datos teórica se incrementa de 11 a 54 Mbps (realmente 24,7 Mbps). Una característica notable de esta normalización es que es compatible con la 802.11b. En el momento de redactar este texto, se está desarrollando la 802.11n, que se espera que alcance los 500 Mbps.

Las redes que trabajan bajo los estándares 802.11b y 802.11g pueden sufrir interferencias por parte de hornos microondas, teléfonos inalámbricos y otros equipos que utilicen la misma banda de 2,4 Ghz.

Bluetooth

Es un estándar de comunicaciones inalámbricas para transmisión de voz y datos entre distintos sistemas utilizando un enlace de radiofrecuencia en el rango de 2,4 a 2,48 GHz y la técnica de espectro expandido. Los saltos de frecuencia se producen entre 79 niveles separados en intervalos de 1 MHz, y con una cadencia máxima de 1.600 saltos/segundo. El estándar define un canal de comunicaciones de tasa de datos máxima de 720 Kbps, con un alcance óptimo de 10 metros y potencia de 1 mW. Existen versiones mejoradas como las Bluetooth 1.0, 1.1 y 2.0, alcanzando esta última una tasa de datos de 2,1 bps.

Se han comercializado circuitos integrados de muy bajo consumo y coste, que incluyen el hardware completo para implementar el estándar Bluetooth. Estos circuitos contienen un subsistema de radio encargado de modular y transmitir la señal, y un procesador digital de señales (DSP) que actúa de controlador del enlace con funciones tales como gestión de protocolos, control de la transferencia (síncrona o asíncrona), codificación de audio y cifrado de datos.

Es una tecnología que se utiliza fundamentalmente para redes de área personal. Los objetivos de la normalización Bluetoth son:

- Facilitar el intercambio de datos entre equipos móviles y fijos.
- Eliminar cables y conectores entre los equipos citados.
- Facilitar la sincronización de datos entre equipos electrónicos de consumo, tales como teléfonos celulares, computadores móviles o de bolsillo, cámaras fotográficas, computadores portátiles, impresoras, equipos de audio, etc.

En la definición de este estándar han participado los principales fabricantes de los sectores de la informática y de las telecomunicaciones, habiéndose adherido posteriormente otros muchos.

ZigBee

Posteriormente se ha desarrollado otro protocolo inalámbrico, muy similar al Bluetooth, denominado **ZigBee**, con el que se consigue aún menor consumo (aunque menor tasa de bits), estando su principal campo de aplicación en edificios inteligentes, domótica y control industrial. Sigue el estándar IEEE 802.15.4 de redes inalámbricas de baja velocidad. La característica más notable de un módulo de comunicaciones ZigBee es su bajo consumo ya que puede ser alimentado con una pila que puede durar de meses a años. La tasa de datos máxima que se consigue es de 250 Kbps en la banda de 2,4 GHz y con un alcance máximo de unos 50 metros que puede variar con las características ambientales (humedad, temperatura y calidad del aire).

El nombre de ZigBee viene de "zigzagueo como una abeja" (*Zigzag like a bee*) y se debe a que estas redes se suelen configurar en forma de malla pudiendo ir las señales de un módulo a otro a través de los módulos intermedios siguiendo diversos caminos (en zigzag, como se desplazan las abejas). Al igual que ocurre con Internet, en caso de que falle un módulo, la comunicación se restablece a través de otro u otros disponibles.

19.1.6 Redes de área amplia (WAN)

Una red de área amplia es toda red que cubre una extensa superficie geográfica (un país o un continente), atraviesa zonas de acceso público y utiliza, en mayor o menor medida, redes de comunicaciones públicas. Por lo general las redes de área amplia son redes en malla, en las que se integran multitud de procesadores de comunicaciones para encaminar los mensajes desde el origen a su destino y controlar el tráfico.

A continuación indicaremos los métodos principales que se usan para compartir líneas (Sección 19.1.6.1) y haremos una referencia a las redes públicas (Sección 19.1.6.2).

19.1.6.1 Métodos para compartir líneas

Se suelen utilizar tres métodos para compartir las líneas:

1. **Conmutación de circuitos** o conmutación de líneas. En el momento de iniciarse la comunicación, o con anterioridad, se asigna un camino determinado a través de la red entre emisor y receptor, este camino queda reservado en exclusiva a ambos mientras dura la comunicación (teléfono, por ejemplo).
2. **Conmutación de paquetes.** Los mensajes se fragmentan en paquetes, que la red de comunicaciones transporta como unidades individuales. Hay dos servicios alternativos en la redes de conmutación de paquetes:
   2.1. Servicio de circuitos virtuales. La red proporciona un canal concreto (*PR-B*, *PR-A, PR-I*, *PR-L* y *PR-G*, por ejemplo, en la Figura 19.2, para conectar *C-A* y *C-L*) identificado por un código (canal 3, por ejemplo). Para establecer la comunicación sólo se necesita la dirección de destino. La red de comunicaciones se encarga del control

de errores y de la regulación del flujo. Los paquetes de información se reciben en el mismo orden en que se despacharon. La selección del camino (canal virtual seleccionado entre todos los posibles) se efectúa al iniciarse la comunicación (encaminamiento estático).

2.2. Servicio datagrama. Los paquetes pueden llegar al equipo terminal de datos receptor, desordenados e incluso entremezclados con los de otros mensajes. Cada paquete necesita incluir su dirección de destino y son tratados individualmente por los procesadores de comunicaciones, donde se almacena temporalmente y se reexpide por el camino más adecuado. La gestión de errores y el control de flujo deben ser hechos por las estaciones terminales. El encaminamiento puede no ser el mismo para los distintos paquetes de un mismo mensaje (encaminamiento dinámico).

3. **Conmutación de mensajes.** Con esta técnica un procesador de red recibe un mensaje, lo almacena hasta comprobar que la ruta adecuada esté libre y entonces lo envía. Este método se suele denominar almacenar y reenviar.

19.1.6.2 Redes públicas de comunicaciones

Hay diversos tipos de redes públicas a través de las cuales se pueden transmitir datos (Tabla 19.4):

Red convencional de audio (analógica)

Para utilizar esta red para transmisión de datos se modula la señal digital con una portadora sinusoidal, de forma que la señal transmitida es analógica. En cada extremo de la línea hay que colocar un **módem analógico** que demodula la señal que recibe y modula la que transmite (Sección 18.2.3.2). La utilización de la red telefónica convencional puede hacerse de dos formas: con líneas conmutadas, en las que para conectar un equipo hay que "marcar" el número correspondiente, y la conexión dura mientras se transmiten los datos, y con líneas punto a punto, en las que las conexiones telefónicas son permanentes. En estas redes la voz se transmite en grupos de canales multiplexados en frecuencia y ocupando cada señal de voz un ancho de banda de 4 KHz, o, alternativamente, en tramas multiplexadas en el tiempo con otras de otros canales.

Tabla 19.4. *Características de algunas redes públicas de transmisión de datos y de métodos de conexión del usuario con la central telefónica.*

Red	Características
Red telefónica convencional (audio)	– Líneas conmutadas (2 hilos), o líneas de uso exclusivo (4 hilos). – Tasas de datos (módem analógico): 300, 1.200, 2.400, 4.800, 9.600 y 56.000 bps
Red digital de servicios integrados: ISDN/RDSI	– RDSI, canales: • B : 64 Kbps; • D1: 16 Kbps; D2: 64Kbps; • H0: 384 Kbps; H1: 1.536Kbps; H2:1.920Kbps – RDSI banda ancha: 155.52 Mbps y 622.08 Mbps
ADSL	– ADSL: • Entrada de datos: 500 Kbps a 8 Mbps; – Salida de datos: 500 Kbps a 1 Mbps – ADSL2: • Entrada de datos: 12 Mbps; • Salida de datos: 1 Mbps – ADSL2+: • Entrada de datos: 24 Mbps; • Salida de datos: 1,2 Mbps
Cable-módem	– Entrada de datos: 3 a 10 Mbps (podría llegar a 36 Mbps) – Salida de datos: 500 Kbps a 1 Mbps (podría llegar a 12 Mbps)

Redes de transmisión de datos (digitales)
La transmisión de datos ha adquirido tal importancia que las compañías telefónicas han creado redes específicas con este objetivo.

Como ejemplo puede citarse la **Red Digital de Servicios Integrados** (**RDSI**, en inglés **ISDN** *Integrated Services Digital Network*) (Tabla 19.4), que es una red conmutada completamente digital y con capacidad multimedia, es decir, que permite ofrecer servicios que van desde la llamada de voz hasta el acceso a redes de información, transmisión de fax a alta velocidad, videoconferencia, etc. La RDSI puede considerarse una red de área amplia que proporciona conectividad universal entre los equipos de los usuarios conectados a ella.

La **RDSI básica** (BRI) ofrece al usuario tres tipos de canales:

- Canal portador (canal B) a 64 Kbps, es el canal básico del usuario y puede transportar cualquier tipo de información en modo dúplex.
- Canal de datos (canal D) de 16 ó 64 Kbps, que usualmente se utiliza como control del canal B.
- Canales híbridos (canales H), que proporcionan velocidades de 384 Kbps (H0), 1.536 Kbps (H11) o 1.920 Kbps. Estos canales se utilizan para transmisión de vídeo, teleconferencias, y otras aplicaciones que requieran tiempo real.

Para interconectar un equipo concreto a una línea RDSI (Figura 19.9) hay que tener en cuenta si dicho equipo sigue la normalización RDSI (*equipos terminales TE1*, como teléfonos y fax digitales) o no (*equipos terminales TE2*, como teléfonos y fax analógicos o computadores). En el segundo de los casos es necesario utilizar un **adaptador de terminal** (*TA*) que es una tarjeta de circuitos que puede instalarse externa o internamente al sistema (computador, por ejemplo). El TA a veces se le denomina **módem RDSI** por conectarse al mismo puerto serie del computador que un módem, y puede incluir la posibilidad de conexión a teléfono y fax analógico. Además la línea de cable telefónico que llega al usuario (abonado) de la red debe conectarse a un **terminal de red** (NT1) que controla las terminaciones físicas y eléctricas de los equipos de los abonados y los conecta a la línea telefónica que llega de la central telefónica.

La red RDSI ofrece los siguientes servicios:

- *Servicios portadores:* se limitan a ofrecer un medio para transferir información (audio, vídeo y datos) a través de la red de forma transparente; es decir, sin modificarlos.
- *Teleservicios:*
  - Telefonía a 7 KHz.
  - Facsímil Grupos 2 y 3 Facsímil Grupo 4.
  - Teletex, videotex, videotelefonía.
- *Servicios suplementarios:*
  - Grupo cerrado de usuarios.
  - Identificación del usuario llamante.
  - Restricción de la identificación del usuario llamante.
  - Identificación de usuario conectado.
  - Restricción de la identificación de usuario conectado.
  - Identificación de llamada en espera.
  - Marcación directa de extensiones.
  - Múltiples números de abonado.

– Marcación abreviada.
– Conferencia a tres.
– Desvío de llamadas.
– Transferencia de llamadas dentro del bus pasivo.
– Información de tarificación. Cobro revertido.

Para cubrir necesidades mayores de 600 Mbps se ha desarrollado la **RDSI de banda ancha**, existiendo comercialmente tres opciones:

- Servicio full-dúplex: a 152,52 Mbps, en ambos sentidos.
- Servicio full-dúplex asimétrico 155,52 Mbps (sentido abonado a red) y 622,08 Mbps (sentido red a abonado).
- Servicio full-dúplex a 622,08 Mbps.

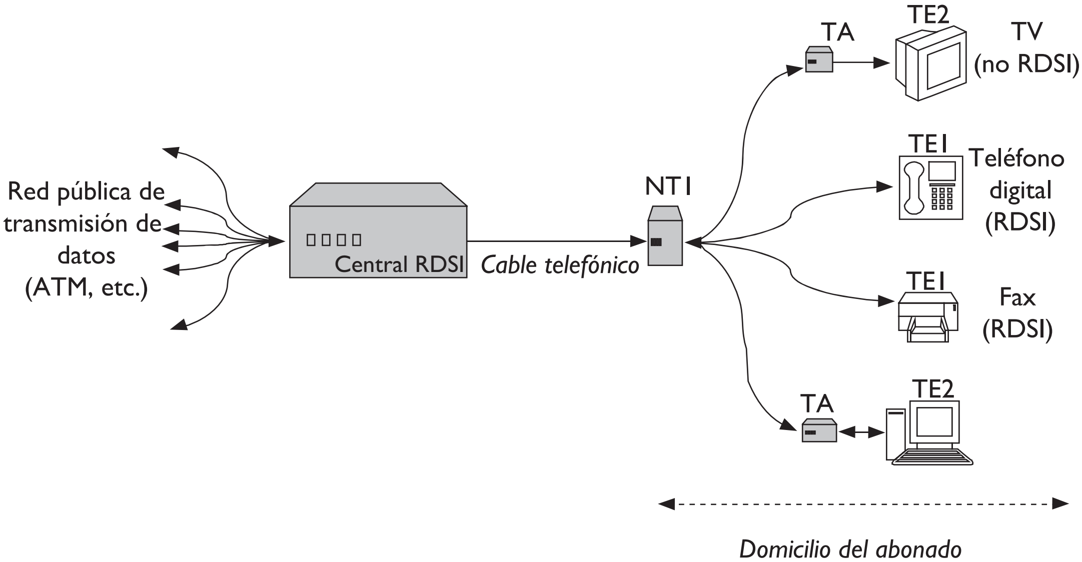


Figura 19.9. *Esquema simplificado de un ejemplo de instalación RDSI en el domicilio del abonado (NT1: Terminal de red, TA: Adaptador de terminal, TE1: Equipo terminal RDSI, TE2: Equipo terminal no RDSI).*

Los servicios portadores que ofrece la RDSI se corresponden a los tres niveles inferiores del modelo OSI, y pueden implementarse utilizando redes de conmutación de circuitos, redes de conmutación de paquetes, de conmutación de tramas (*Frame relay*) o de conmutación de celdas (ATM). Tiene especial interés el último método: **conmutación de celdas**, también denominado **modo de transmisión asíncrono** (**ATM** *Asynchronous Transfer Mode*). Se fundamenta en la idea de que la infraestructura de comunicación de datos se ha perfeccionado notablemente y que por tanto no es necesaria la inclusión de tantos controles en las cabeceras de los paquetes y tramas como se hace en los paquetes tradicionales. La reducción de este tipo de información hace que la velocidad efectiva de transmisión de los datos de los usuarios pueda ser mucho mayor (véase Ejercicio 19.3). En ATM las unidades de transmisión (paquetes) se denominan **celdas** y son de tamaño fijo y relativamente pequeño (53 octetos, 5 de ellos de control), siendo mucho más fáciles y rápido de tratar por los procesadores de red que los

paquetes. De esta forma se consiguen velocidades de transmisión muy elevadas (de 50 Mbps a 10 Gbps), permitiendo la transmisión a través de la misma red de señales de voz, audio, datos y vídeo. Piénsese que para obtener una adecuada calidad y la visión de imágenes animadas, como es el caso de la televisión de alta definición (**HDTV**, *High Definition TV*) se requiere una tasa de datos de 19,4 Mbps (Figura 18.1).

Existen otras alternativas de conexión a las redes públicas de transmisión de datos, como son la DSL y la de cable coaxial (cable módem), que a continuación se describen brevemente.

DSL *(Digital Subscriber Line*, línea de abonado digital)
Esta tecnología incrementa considerablemente la capacidad de transporte de información digital en las líneas locales (pares trenzados sin apantallar) que comunican a los abonados con las centrales de la red telefónica. La idea básica es utilizar la multiplexación en frecuencia (Sección 18.2.4.1) junto con sofisticados métodos de modulación para transmitir a través del mismo medio señales de voz (telefonía normal) y datos digitales a velocidades notables. Hay dos categorías: DSL simétrica y DSL asimétrica o ADSL *(Asymetric Digital Subscriber Line).* En el primer caso (SDSL, HDLS, etc.) las conexiones se realizan a igual velocidad en los dos sentidos: usuario a central y central a usuario; no ocurriendo lo mismo en ADSL. El fundamento del concepto ADSL radica en que se ha observado que la mayor parte de los usuarios utilizan aplicaciones que requieren menor velocidad de transmisión en el sentido usuario a red que de la red al usuario. Este es, por ejemplo, el caso del acceso a páginas web. El PC del usuario tiene que proporcionar muy poca información (textos) para poder acceder a una página determinada; pero, sin embargo, él debe obtener mucha información (la página web, con imágenes, etc.) en tiempos reducidos. Teniendo en cuenta este hecho en ADSL asigna un ancho de banda de 0 a 1 MHz de la siguiente forma:

- 0 a 25 KHz: transmisión de voz (telefonía normal).
- 25 a 200 KHz, comunicación de datos de usuario a red de 500 Kbps a 1 Mbps (según define el estándar ANSI).
- 200 a 1.000 KHz, comunicación de datos de red a usuario de 500 Kbps a 8 Mbps (según define el estándar ANSI).

De esta forma se aprovecha muy eficientemente el ancho de banda de acuerdo con las necesidades del usuario. Hay otras variantes de DSL, como son:

- **ADSL2** y **ADSL2+:** con capacidad de proporcionar un servicio, además de datos, de televisión y vídeo de alta calidad a través de la línea de teléfono convencional (par de hilos de cobre).
- **RADSL:** línea de abonado digital con velocidad adaptativa, de acuerdo con la naturaleza de la información a transmitir (audio, datos, multimedia, etc.) y las necesidades de los usuarios.
- **HDSL:** línea de abonado digital de alta velocidad, que consigue tasas de datos de hasta 2 Mbps a distancias de 3,6 Km sin repetidores.
- **IDSL:** línea de abonado digital RDSI.
- **SDSL:** línea de abonado digital simétrica.
- **VDSL:** línea de abonado digital de muy alta velocidad, con tasas de datos de entrada de 50 a 55 Mbps y de salida de 1,5 a 2,5 Mbps.

En la Figura 19.10 se muestra un esquema de los componentes que requiere un ejemplo de instalación ADSL en el domicilio del usuario. El microfiltro se utiliza para eliminar las frecuencias que no corresponden a la señal de voz propia del teléfono.

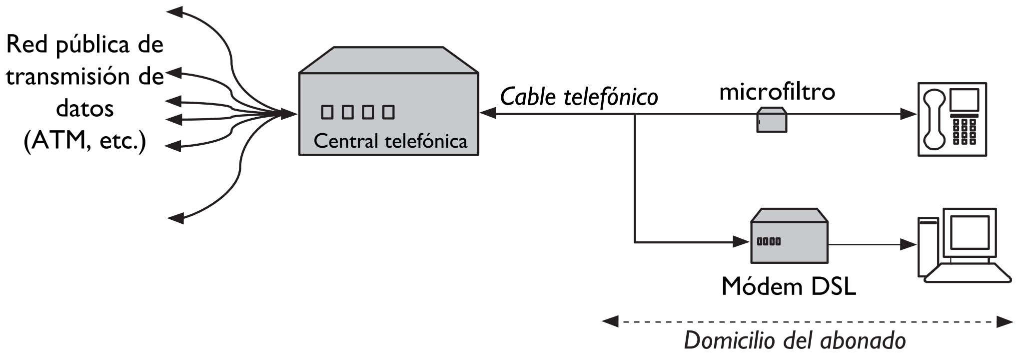


Figura 19.10. *Esquema simplificado de un ejemplo de instalación ADSL en el domicilio del abonado.*

Cable módem

Es una tecnología que permite conectarse a una red utilizando cable coaxial de antena de TV, consiguiéndose así velocidades más altas que con una línea convencional. Trata de aprovechar el gran ancho de banda que presenta el cable coaxial y la infraestructura que hay en muchas ciudades de **televisión por cable** (**CATV**); así resulta posible a través de estas líneas ofrecer a los abonados simultáneamente televisión, teléfono y acceso a Internet (datos). Es una alternativa a RDSI o ADSL. Los abonados de un mismo vecindario comparten por *multiplexación en el tiempo* (Sección 18.2.4.3) el ancho de banda proporcionado por una única línea de cable coaxial. Por lo tanto, la velocidad de conexión puede variar dependiendo de la cantidad de equipos que estén usando el servicio al mismo tiempo. Teóricamente se pueden llegar a conseguir velocidades de hasta 27 Mbps en recepción de datos. En la Figura 19.11 se muestra un ejemplo de una instalación de cable-módem en el domicilio de un usuario.

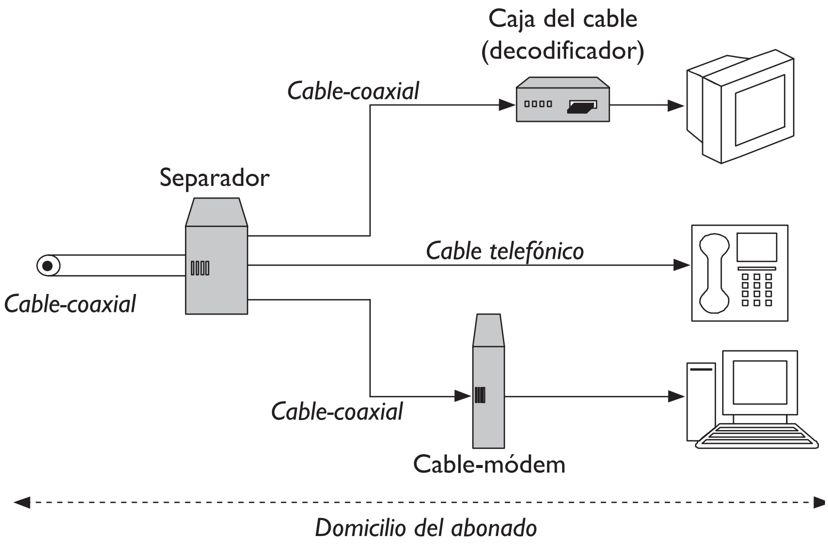


Figura 19.11. *Esquema simplificado de un ejemplo de instalación cable-módem en el domicilio del abonado.*

19.2 Internet

En las secciones anteriores hemos considerado distintos tipos de redes, como las LAN y las WAN. Ahora bien, diversas redes individuales se pueden interconectar entre sí formando una interconexión-de-redes (*internetwork*) o, abreviadamente, una **internet**. Por otra parte, se denomina **Internet** (con mayúscula la primera letra) a la red formada por la interconexión de redes a lo ancho de todo el mundo.

19.2.1 Dispositivos de interconexión de redes

Existen diversas formas de ampliar una red aislada o interconectar redes individuales, con el objetivo de compartir los computadores y los recursos que contienen. Estos sistemas se enumeran en la Tabla 19.5 y se describen brevemente a continuación (véase Figura 19.12).

Tabla 19.5. *Dispositivos de interconexión de redes.*

– Repetidor *(repeater)*
– Concentrador (*hub*)
– Puente *(bridge)*
– Conmutador *(switch)*
– Dispositivos de encaminamiento *(router)*
– Pasarela *(gateway)*

- **Repetidor** (*Repeater*). Es un dispositivo electrónico que conecta dos segmentos de una misma red, transfiriendo el tráfico de uno a otro. Los segmentos de red suelen tener limitada su longitud, debido a la atenuación de las señales y al ruido que se puede inducir en ellas. Con un repetidor se puede evitar el problema de la longitud, ya que reconstruye las señales que recibe de cada uno de los segmentos y las transmite al otro. Un repetidor, por tanto, actúa sólo en el nivel físico del modelo OSI.
- **Concentrador** (*Hub*). Contiene diversos puntos de interconexión, denominados puertos, retransmitiendo cada paquete de datos recibido por uno de los puertos a los demás puertos. Se utiliza para implementar redes con topología en estrella. Actúa en el nivel físico del modelo OSI.
- **Puente** (*Bridge*). Como los repetidores, permiten conectar *dos* segmentos de red, pero, a diferencia de ellos, seleccionan el tráfico que pasa de un segmento a otro, de forma tal que sólo el tráfico que parte de una estación de un segmento y que va al otro segmento se transfiere a través del puente. Con un puente, por tanto, se puede reducir notablemente el tráfico en los distintos segmentos conectados a él. Los puentes actúan en el nivel físico y de enlace de datos del modelo OSI (capa 2). En el nivel de enlace, el puente comprueba la dirección de destino y hace la copia hacia el otro segmento si allí se encuentra la estación de destino. La principal diferencia con un repetidor es que éste hace pasar todas las tramas que le llegan al otro segmento, independientemente de que se encuentre o no allí la estación de destino.
- **Conmutador** (*Switch*). Interconecta dos o más segmentos de red, pasando segmentos de uno a otro de acuerdo con la dirección de control de acceso al medio (MAC). Actúan como filtros, en la capa de enlace de datos (capa 2) del modelo OSI. Las funciones, por lo tanto, son iguales que las de un puente, pero pudiendo interconectar y filtrar la información entre más de dos redes. La principal diferencia con un concentrador, es que

éste hace pasar todas las tramas que le llegan a los otros puertos, independientemente de que se encuentre o no allí la estación de destino.

- **Dispositivo de encaminamiento** (*router*). Operan entre redes aisladas que utilizan protocolos similares y direccionan o encaminan la información de acuerdo con la mejor ruta posible en un momento dado. Un mismo dispositivo de encaminamiento está conectado a dos o más redes, e implica la realización de tareas que conciernen a los tres niveles inferiores del modelo OSI: físico, enlace de datos y red.
- **Pasarela** (*Gateway*). Son dispositivos de encaminamiento que contienen programas adicionales (correspondientes a los niveles de transporte, sesión, presentación y aplicación, del modelo OSI), que permiten interconectar redes que utilizan distintos protocolos: por ejemplo TCP/IP, SNA, NetWare y AppleTalk. Las pasarelas deben desensamblar las tramas y paquetes que le llegan para obtener el mensaje original y a partir de éste volver a reconfigurar los paquetes y las tramas, pero de acuerdo con el protocolo de la red donde se encuentra la estación de destino (Figura 19.26).

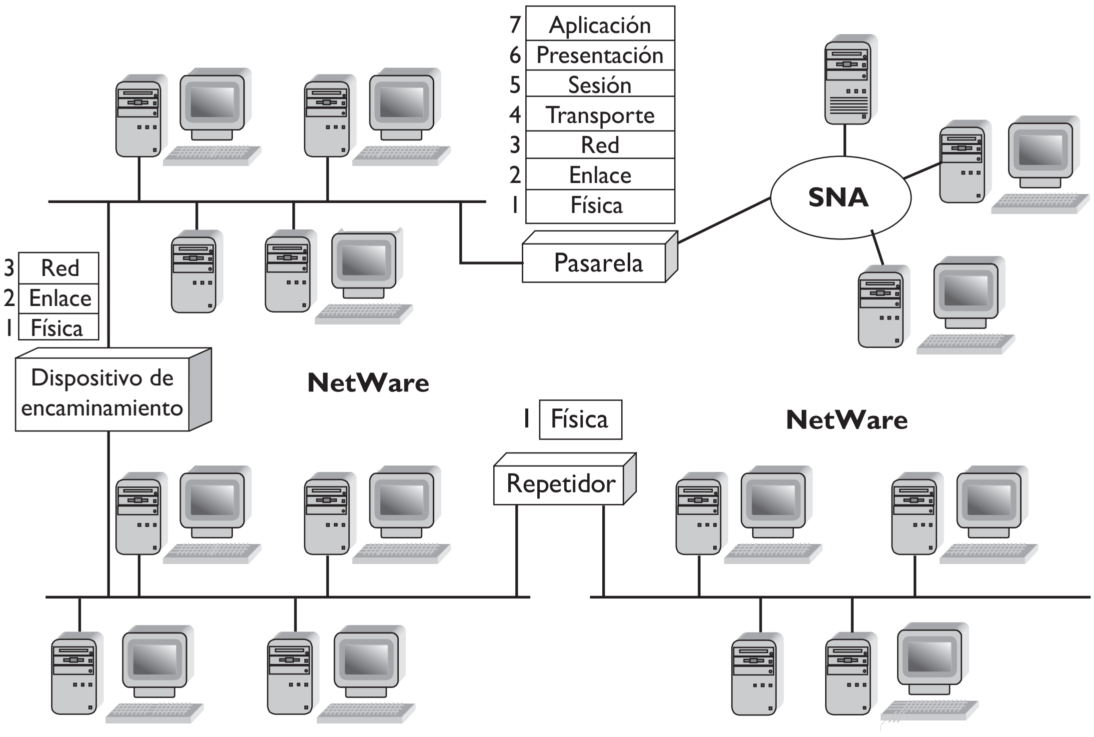


Figura 19.12. *Redes interconectadas con un repetidor, un dispositivo de encaminamiento y una pasarela*

Internet está formada por múltiples redes interconectadas por medio de dispositivos de encaminamiento y pasarelas. Bajo el punto de vista del usuario de Internet, las estaciones de la red son computadores (independientemente de su capacidad), cada uno de ellos identificado por una dirección, y la información se transmite de acuerdo con el conjunto de protocolos TCP/IP. El protocolo TCP/IP hace abstracción de las redes individuales y considera a todos los computadores de la red como si estuviesen conectados a una red única. Con precisión podríamos definir

Internet como *la red formada por la interconexión de redes a lo ancho de todo el mundo que utiliza el protocolo TCP/IP.*

Una empresa u organización puede disponer de varias redes particulares y desear que los distintos computadores se intercomuniquen unos con otros utilizando las mismas posibilidades que se tienen con Internet. Un conjunto de redes de este tipo se denomina **intranet** y suele estar conectado también a Internet, pero a través de un **cortafuegos** (Figura 19.13). Un cortafuegos es sencillamente una pasarela o un computador con el objetivo adicional de reforzar la seguridad, controlando los accesos externos a la información y servicios ofrecidos por la intranet (examinando y filtrando los paquetes, por ejemplo).

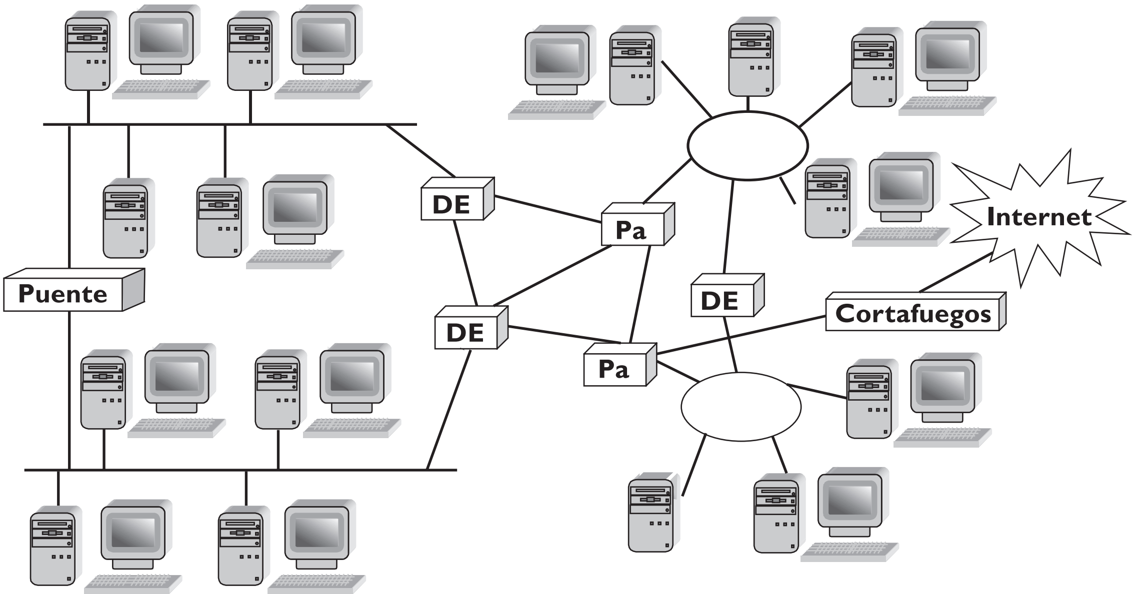


Figura 19.13. *Ejemplo de intranet.*

19.2.2 Direccionamiento en Internet

En Internet un computador individual se identifica por medio de una **dirección IP** (*IP: Internet protocol*), que está formada por dos partes: el código identificador de la red y el código identificador del computador en la red. La dirección IP esta formada en total por 4 Bytes, que se suelen dar cada uno de ellos en decimal, separados por puntos (**notación punto decimal**). El número de bytes que identifica la red puede ser de 1, 2 ó 3 Bytes (redes de tipo A, B y C, respectivamente, véase Ejercicio 19.12) dependiendo de que el número de computadores conectados a ella sea mayor o menor, respectivamente; pero siempre la dirección IP se compone de 4 Bytes.

EJEMPLO 19.4

La dirección IP 150.214.191.37 consta del identificador de red 150.214.191, que identifica un dominio de la Universidad de Granada, y 37, que se refiere a un computador en concreto.

EJEMPLO 19.5

Obtener la dirección IP en decimal correspondiente a la dirección binaria:

1001 0110 0110 1101 0101 1011 0010 0101

Primero formamos cuatro grupos con los bits:

1001 0110. 0110 1101. 0101 1011. 0010 0101

y ahora pasamos a decimal cada grupo:

1001 0110 = 150

0110 1101 = 214

0101 1011 = 191

0010 0101 = 37

Luego, la dirección IP en notación decimal es: ***150.214.191.37***

Si un organismo desea incorporar a Internet sus computadores, debe designar una persona responsable de la red y ésta tiene que solicitar al *Internet Network Information Center* un conjunto de direcciones IP que identifiquen la nueva red. Por otra parte, el responsable de la red se encarga de asignar a cada computador concreto su dirección IP. Así, la dirección IP de un PC del Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Granada es 150.214.103.79.

Tabla 19.6. *Algunos de los códigos en los que se agrupan los nombres de dominios de Internet.*

Tipo de actividad	Tipo de dominio
Arte	.arts
Comercial	.com
Educación	.edu
Negocios	.firm
Gubernamental	.gov
Servicios de información	.info
Militar	.mil
Servicios de la red	.net
Particulares y familias	.nom
Organización genérica, sin ánimo de lucro	.org
Servicios de la red	.rec
Tiendas y almacenes	.store
Entidades relacionadas con web	.web

Estado	Tipo de dominio
Argentina	.ar
Bolivia	.bo
Brasil	.br
Costa Rica	.cr
Cuba	.cu
Chile	.cl
Colombia	.co
Ecuador	.ec
El Salvador	.sv
España	.es
Guatemala	.gt
Honduras	.hn
Nicaragua	.ni
Panamá	.pa
Paraguay	.py
Perú	.pe
Puerto Rico	.pr
Uruguay	.uy
Venezuela	.ve

Las direcciones IP son numéricas y por tanto poco inteligibles para los humanos. Debido a ello, el InterNIC también proporciona para cada red o subred un **nombre de dominio** único que es un nemónico. Así el dominio de la Universidad de Granada tiene de nombre *ugr.es*. Los caracteres después del último punto identifican el tipo de dominio de que se trata, de acuerdo con el significado que se da en la Tabla 19.12. También las máquinas individuales disponen de un nombre simbólico; así, atcpc27.ugr.es es el nombre simbólico del PC del Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Granada citado anteriormente. Hay servidores de nombres de dominio (**DNS**, *Domain Name Server*) que proporcionan la dirección IP asociada a un nombre de dominio dado.

En realidad un mismo computador de usuario o de red puede tener más de una dirección IP, ya que, rigurosamente hablando, la dirección IP corresponde a la *interfaz con la red*. Éste es el caso de los dispositivos de encaminamiento, que tendrán tantas direcciones IP como conexiones a red.

19.2.3 Protocolos TCP/IP

El conjunto de protocolos de comunicaciones utilizados por Internet (TCP/IP) fue desarrollado por el Departamento de Defensa de Estados Unidos con el objetivo de distribuir geográficamente los recursos de cómputo e interconectarlos adecuadamente de forma que la destrucción por un ataque militar de algún computador o enlace de la red no paralizase el sistema global. Estos protocolos en la actualidad están considerados *de facto* como un estándar muy eficiente y flexible para intercomunicar computadores heterogéneos. Realmente TCP (*Transmission Control Protocol*) e IP (*Internet Protocol*) son únicamente el nombre de dos de los protocolos del conjunto, donde se incluyen otros más como UDP (*User Datagram Protocol*), HTTP (*Hypertext Transfer Protocol*), SMTP (*Simple Mail Transfer Protocol*), ICMP (*Internet Control Message Protocol*), RTP (*Real Time Transport Protocol*) y FTP (*File Transfer Protocol*).

Su concepción se inició antes del desarrollo del modelo OSI y no sigue las sugerencias de este último. Así, en vez de ser un modelo por capas es un *modelo jerárquico*, en el que una entidad puede utilizar servicios de cualquier capa inferior, y no sólo de la inferior adyacente. En el conjunto TCP/IP se consideran cuatro niveles, cuya relación con el modelo OSI se indica en la Figura 19.14a y describimos brevemente a continuación.

	OSI	TCP/IP	
7	Aplicaciones	Aplicaciones	Mensaje
6	Presentación		
5	Sesión		
4	Transporte	Transporte	Segmentos o datagramas de usuario
3	Red	Internet	Paquetes IP
2	Enlace de datos	Interfaz con la red	Tramas
1	Física	Física	bits

(a)

Figura 19.14. *Protocolos TCP/IP. (a) Relación con el modelo OSI.*

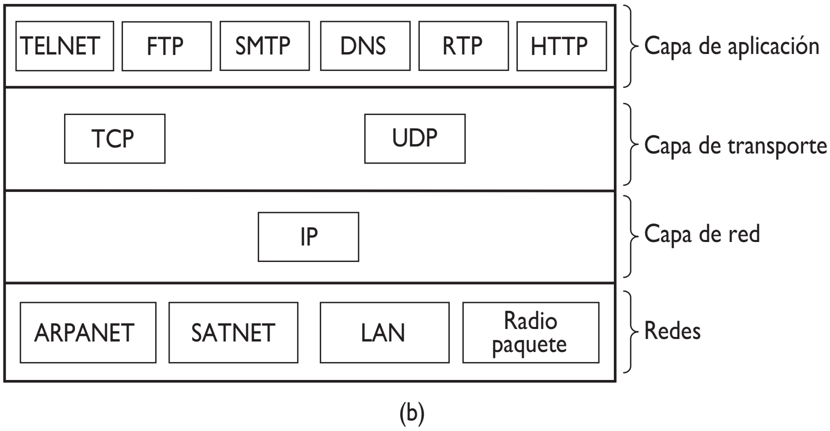


Figura 19.14. *(Cont.) Protocolos TCP/IP; (b) Interrelación entre los distintas capas.*

Nivel de procesos y aplicaciones

Proporciona la comunicación entre procesos o aplicaciones de computadores separados, suministrando servicios que pueden ser usados por otras aplicaciones, tales como las que se corresponden a los siguientes protocolos:

- **SMTP:** para correo electrónico (SMTP, *Simple mail transfer protocol*).
- **FTP:** transferencia de archivos (FTP, *File transfer protocol*).
- **TELNET:** conexión de terminal remoto o terminal virtual (TELNET).
- **HTTP:** envío de mensajes usando TCP.
- **DNS y RTP:** envío de mensajes utilizando UDP.

La interacción entre aplicaciones se realiza usualmente utilizando el **modelo cliente/servidor**. Según este modelo un proceso de una de las estaciones finales (*cliente*) solicita a otro proceso de la estación en el otro extremo (*servidor*) un servicio. El servidor suministra las respuestas a los requerimientos del cliente (Figura 19.15). El protocolo HTTP, por ejemplo, especifica las reglas de cómo el cliente y el servidor interactúan para recuperar un documento: el cliente solicita la visualización de una determinada página web y el servidor se la proporciona o da los mensajes de error oportunos.

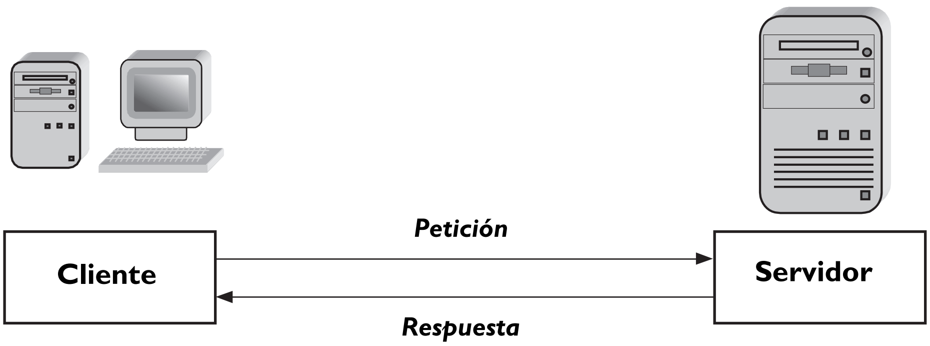


Figura 19.15. *Modelo cliente-servidor, usado en los protocolos de aplicación del conjunto TCP/IP.*

La capa de aplicación se corresponde con las tres capas superiores del modelo OSI: aplicación, sesión y presentación; y, según indicamos anteriormente, en el conjunto TCP/IP las aplicaciones no siempre interactúan con la capa inmediata inferior (transporte o TCP), sino que también pueden relacionarse directamente con las capas de red (IP), e incluso con la capa de interfaz con la red.

Nivel de transporte (control de transmisiones)
El conjunto de protocolos TCP/IP contiene dos protocolos alternativos para implementar el nivel de transporte: el **TCP** (**Protocolo de control de transmisión** o *Transmission Control Protocol*) y el **UDP** (**Protocolo de datagrama de usuario** o *User Datagram Protocol*).

El nivel de transporte recibe de la capa de aplicaciones unidades de información, denominadas **mensajes**, y a partir de ellas, genera **segmentos** o **datagramas de usuario**, dependiendo de si se utiliza TCP o UDP, respectivamente.

El protocolo TCP, antes de enviar datos envía un segmento de control a la capa de transporte de la estación de destino indicando que se disponga a recibir datos para una aplicación (proceso) determinada de la capa de aplicación y otras variables que son usadas por el protocolo; después espera a recibir un segmento de reconocimiento de la estación destino antes de enviar los segmentos de datos. Este tipo de funcionamiento, según el cual antes de enviar los datos se abre la comunicación, se denomina **orientado a conexión**. Descompone los mensajes en segmentos (de 1 a aproximadamente 1.500 caracteres de capacidad), que numera adecuadamente con objeto de que el protocolo TCP del receptor pueda recomponer el mensaje original. En cada segmento se incluye una cabecera con campos que identifica el proceso de la capa de aplicación que lo envía, el proceso de la capa de aplicación que lo debería recibir y da información para la detección de errores (Sección 19.2.6). Por otra parte el protocolo TCP de las dos estaciones se encarga de regular el flujo de información, asegurando que los segmentos lleguen sin error y en secuencia, generando acuses de recibo, y retransmitiendo los paquetes que eventualmente se pierdan. Por este motivo se dice que TCP es un **protocolo fiable**. El nivel TCP del receptor comprueba si los segmentos llegan intactos, los reensambla reconstruyendo así el mensaje original y suministra éste a la aplicación destino.

UDP es un protocolo mucho más sencillo que TCP, de tipo extremo-a-extremo, sin conexión, y no fiable. Antes de enviar datos no se establece una conexión, sino que meramente envía datos (datagramas de usuario) a la dirección de destino, incluso aunque la estación de destino no esté funcionando. UDP tampoco se encarga de la confirmación de la adecuada recepción de los datagramas de usuario; utilizándose cuando la fiabilidad y seguridad son menos importantes que la velocidad.

Protocolo inter-redes o internet (IP: Internet Protocol)
La capa internet gestiona la transferencia de información (segmentos o datagramas) a través de múltiples redes por medio del uso de dispositivos de encaminamiento (o pasarelas).

La capa IP del computador origen encapsula datagramas de usuario o segmentos recibidos de la capa de transporte en **paquetes** (también denominados **datagramas IP**). La cabecera de cada paquete contiene diversos campos con información tal como: dirección IP de origen, dirección IP de destino, longitud total del paquete, número de secuencia del paquete, protocolo de la capa de transporte (TCP o UDP), tiempo de vida y suma de comprobación. La longitud total del paquete puede llegar hasta 64 KB (con el protocolo IPv6 este valor se incrementa hasta 4 GB), aunque con frecuencia no se llegan a estos valores máximos por no admitir la red física de transporte valores tan altos (por ejemplo, Ethernet 1.500 Bytes). El campo **tiempo de vida**, en la versión IPv6, actúa como un contador de saltos, ya que cada vez que un paquete es reexpedido

de un dispositivo de encaminamiento a otro (es decir, cada vez que se produce un *salto*) su valor se decrementa en una unidad, y, si llega a cero, el paquete se destruye, informándose al emisor de este hecho. Con ello se evita que existan paquetes perdidos deambulando indefinidamente a través de internet. Usualmente un valor inicial de 64 de tiempo de vida es más que suficiente para que la mayoría de paquetes lleguen a su destino.

La capa IP de los dispositivos de encaminamiento actúa conjuntamente para encaminar los paquetes desde el origen al destino (de acuerdo con las direcciones IP), manteniendo una tabla de ruta que utiliza para determinar como enviar cada paquete IP. Cuando un paquete IP llega a un dispositivo de encaminamiento los procesos de su capa IP se encargan de analizar los distintos campos de la cabecera y verificar si es correcta la suma de comprobación que se incluye en la propia cabecera, actualizar los campos de la cabecera que lo requieran (por ejemplo, el de tiempo de vida y la suma de comprobación), y, a partir de la dirección destino y consultando las tablas de rutas, establecer la línea por donde se debe reexpedir el paquete para acercarse a su destino.

Capa de interfaz de red

Es responsable de aceptar paquetes IP y transmitirlos a través de una red específica de comunicaciones. Existen diversas interfaces disponibles para redes de tipos tales como X.25, ATM, retransmisión de tramas, Ethernet y anillo con paso de testigo. Con esta capa se independizan las funciones de la capa internet y las superiores de los detalles de la red subyacente.

19.2.4 Aplicaciones de Internet

Esta sección se dedica a describir brevemente algunas de las aplicaciones de Internet que se enumeran en la Tabla 19.7. La Sección 19.3 se dedica exclusivamente a la Web. El significado de las siglas de los protocolos o utilidades que se nombran se da en la tabla citada.

Tabla 19.7. *Algunas aplicaciones de Internet y sus respectivos protocolos.*

Aplicaciones de Internet	– *Aplicaciones básicas*	• Terminal virtual (protocolo TELNET). • Transferencia de ficheros (protocolo FTP, *File Transfer Protocol*). • Sistema de ficheros en red (protocolo NFS, *Nework File System*). • Correo electrónico (*e-mail*, protocolo SMTP, *Simple Mail Transfer Protocol*). • Tertulias Internet o *chats* (IRC, *Internet Relay Chat*). • Búsqueda de archivos: Archie, Gopher, Verónica, WAIS (Servicio de información de área mundial o *Wide Area Information Service*).
	– *Aplicaciones multimedia en Internet*	• Videoconferencia. • Vídeo bajo demanda. • Voz a través de IP (VoIP, *Voice over IP*).
	– *Web (WWW, World Wide Web)*	• Transferencia de hipertextos (protocolo HTTP, Hipertext Transfer protocol).

Conexión de terminal remoto (TELNET)

Posibilita a un usuario conectarse a cualquier computador remoto en cualquiera de las redes interconectadas utilizando el conjunto de protocolos TCP/IP. La conexión se hace emulando, en el

equipo del usuario, un **terminal virtual**, de forma que puede utilizar el computador o terminal como si dispusiese de un terminal *conectado directamente* al computador del otro extremo. El protocolo define las características de un terminal normalizado TCP/IP (el terminal virtual). La aplicación TELNET se compone de dos módulos; el primero de ellos transforma las particularidades del terminal o computador local en las del terminal virtual, y viceversa (para el caso de recepción de información), y el segundo se encarga de interactuar con la aplicación dando la impresión a ésta de que actúa con un terminal local. TELNET también se encarga de solicitar una conexión TCP, a través de la cual se efectúan todos los intercambios de información.

Transferencia de archivos

Para transferir archivos a través de Internet se suele utilizar el **Protocolo para transferencia de ficheros** (FTP), que sirve para transmitir archivos de un computador a otra a petición del usuario (cliente), que dispone de un lenguaje de órdenes para este objetivo. Esta operación se hace utilizando requiere la identificación y palabra clave del usuario, antes de la transferencia del archivo. Por otra parte se establecen dos conexiones TCP. La primera, *conexión de control*, sirve para intercambiar información de autentificación del usuario, así como para la identificación del archivo a transferir y los lugares (directorios o carpetas) de origen y de destino. Después de establecerse la conexión de control, se realiza la segunda conexión, *conexión de datos* a través de la cual se efectúa la transferencia del archivo, que puede contener tanto textos como información binaria. La conexión de control permanece abierta, indicando al usuario, concurrentemente con la transferencia de datos, el número de bytes transmitidos, tiempo estimado para concluir la transferencia, etc. y para, una vez finalizada la transmisión, poder realizar nuevas transferencias.

Debido a que las comunicaciones no suelen ir cifradas en el proceso de envío, por lo que actualmente se utilizan aplicaciones tales como **SSH** (*Secure Shell*) que añaden las técnicas necesarias para establecer la comunicación de forma segura.

Sistema de ficheros en la red (NFS)

El sistema de ficheros en la red (NFS) es un protocolo que permite acceder al cliente a un archivo remoto, como si estuviese en la máquina del cliente, de forma que puede editarlo, o incluso transferirlo, directamente a una tercera máquina.

Correo electrónico

El correo electrónico tal vez sea el servicio más utilizado en Internet, ya que con él un usuario puede enviar muy fácilmente mensajes o archivos a otro computador. El sistema se fundamenta en la utilización del concepto de **buzón** y de **dirección de buzón**. El buzón se asocia a una persona o ente determinado. Una dirección de un buzón se compone de dos partes diferenciadas y separadas por el carácter @.

Nombre–local–del–buzón @ Nombre–del–dominio

La segunda parte corresponde al nombre del dominio del destino y la primera a la dirección del buzón dentro del dominio; así *sprieto@atc.ugr.es,* es el buzón de una persona (Susana Prieto) dentro del subdominio *atc* del dominio *ugr.es.*

Es posible hacer llegar un mismo mensaje a varios buzones, o bien especificando en el mismo correo cada una de las direcciones de destino, o bien utilizando **listas de correo**, a las que se les asocia un nombre. También se puede asignar a una dirección o nombre de listas nombres alternativos, denominados *alias,* así a la dirección *sprieto@ugr.es* le podríamos asociar el alias de *suki,* con lo que para enviar mensajes a ese buzón utilizaríamos el nombre *suki* en lugar de la dirección completa. A un mensaje de correo electrónico se le pueden adjuntar archivos de texto, sonido o imágenes que se transmiten junto con el mensaje.

El correo electrónico en Internet se implementa con el **Protocolo sencillo de de transferencia de correo**, **SMTP** (*Simple Mail Transfer Protocol*), que esta formado básicamente (Figura 19.16) por:

- Un **agente de usuario de correo**, **MUA** (*Mail User Agent*), que se implementa por medio de la aplicación (*Eudora*, *Outlook*, etc.) que se encarga de preparar el mensaje, crear el "sobre" y colocar el mensaje dentro del sobre, así como de visualizar los mensajes que llegan. Además de una interfaz de usuario, incluye otras utilidades para facilitar el trabajo, como puede ser una agenda con listas de direcciones, firma común para todos los mensajes, etc.
- Un **agente de transferencia de correo**, **MTA** (*Mail Transfer Agent*), que se encarga de transferir el mensaje a través de Internet abriendo una sesión TCP de transferencia de archivos.

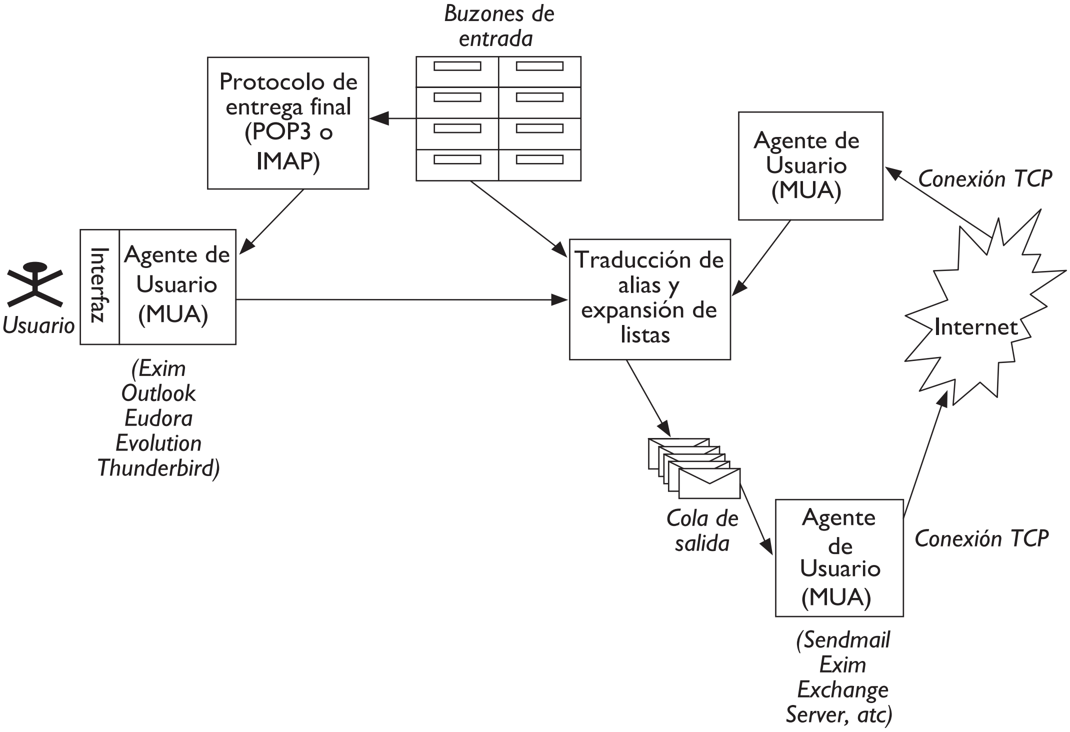


Figura 19.16. *Esquema del funcionamiento del correo electrónico.*

Cuando un usuario crea un mensaje utiliza el agente de usuario, éste mismo lleva el mensaje a un módulo de traducción de alias y expansión de listas de correo y coloca todos los mensajes generados, con sus direcciones completas, en una cola de salida (Figura 19.16). El agente de transferencia, toma los correos de la cola de salida y, de acuerdo con la dirección del sobre, los transmite estableciendo como cliente las conexiones TCP oportunas con los servidores de destino.

El proceso de llegada de un mensaje es análogo, actuando el equipo receptor como servidor y el emisor como cliente. Cuando llega un mensaje, el agente de transferencia de mensajes, lo lleva al módulo de traducción de alias y, si hay algún mensaje a reexpedir, lo incluye en la cola de salida, si no es así lo lleva al correspondiente buzón de entrada. El agente de usuario consulta

periódicamente el buzón de entrada y, si detecta algún correo nuevo, informa al usuario a través de la interfaz correspondiente.

El protocolo SMTP presenta el inconveniente de que sólo es capaz de gestionar y transmitir texto ASCII (7 bits). Para poder transmitir cualquier tipo de información (multimedia, por ejemplo) se ha desarrollado una ampliación denominada **MIME** (**ampliaciones multipropósito de correo de Internet**, *Multi-Purpose Internet Mail Extension*) que transforma, en el emisor, los datos del mensaje no ASCII en datos ASCII en formato **ASCII-NVT** (**Terminal Virtual de Red**). De esta forma el protocolo SMTP del destino (servidor) sólo recibe caracteres ASCII, que pasa al módulo NIME del receptor para la recuperación de la información en su forma original.

Con frecuencia el usuario de correo electrónico no tiene permanentemente conectado su PC a la red, con lo que si otra persona le envía un mensaje no se puede establecer la conexión TCP. Para solucionar este problema lo usual es disponer de acceso a un servidor de correo electrónico que esté conectado permanentemente a Internet y se encargue de recibir los mensajes. Se han establecido **protocolos de entrega final** que, al activar el usuario la aplicación del correo, consultan periódicamente el buzón al servidor de correo, y, en su caso, descargan sobre el PC los correos del buzón correspondiente (Figura 19.16). Los protocolos de entrega final más conocidos son el **POP3** y el **IMAP**; cuando el primero de ellos hace la descarga de correos en el equipo del usuario final, elimina los mismos del servidor, mientras que con el segundo de ellos se mantienen durante un tiempo predeterminado o hasta cuando explícitamente se borren.

Tertulias Internet o *chats* (IRC)

El IRC o *chat* es una aplicación mediante la cual distintos usuarios de Internet pueden comunicarse entre sí en tiempo real. La diferencia fundamental con el correo electrónico reside en que la comunicación es instantánea, pudiendo intercomunicarse por escrito unos con otros en el mismo momento.

Aplicaciones de búsqueda de archivos

Existe una serie de aplicaciones, entre las que se encuentran **Archie**, **Gopher**, **Verónica** y **WAIS** (**Servicio de información de área mundial**), cuyo objetivo es buscar archivos a través de Internet que traten sobre un determinado tema. La localización de los archivos se realiza, dependiendo del programa, buscando palabras clave en el título del mismo o dentro de su contenido; por lo general, las búsquedas pueden perfilarse adecuadamente utilizando los operadores básicos del álgebra de conjuntos (unión, intersección y negación) para combinar las palabras clave de la búsqueda. En la actualidad estos programas están en desuso ya que han sido superados ampliamente por los **motores** o **robots de búsqueda** utilizados dentro de la web (Sección 19.3.3).

Aplicaciones multimedia en Internet

En este grupo se incluyen aplicaciones tales como videoconferencia, transmisión de audio o vídeo bajo demanda realizada utilizando el protocolo internet (IP). Este conjunto de aplicaciones se caracteriza por las necesidades de utilizar grandes anchos de banda y realizar ejecuciones en tiempo real, con retardos muy bajos y una alta calidad. Utilizando técnicas de compresión de datos, como las descritas en la Sección 4.7, se consigue una reducción substancial de los anchos de banda inicialmente requeridos.

19.3 La web

La red mundial *World Wide Web* (abreviadamente **www** o sencillamente **web**) es un conjunto de documentos (archivos) distribuidos a lo largo de todo el mundo y con enlaces entre ellos.

Los documentos se visualizan con palabras, frases o imágenes resaltadas (en negrita o en un color determinado) debajo de las cuales se ocultan punteros a otros documentos ubicados en el propio servidor u otro. Seleccionando y activando con el ratón un elemento con puntero, automáticamente se hace una llamada al documento apuntado.

EJEMPLO 19.6

Suponga que se obtiene a través de la web el siguiente fragmento de texto:

> *La **Costa Tropical** fue ayer escenario de un acontecimiento importante, la inauguración de un establecimiento hotelero de la **cadena Sol Meliá.** Clasificado en la categoría de cuatro estrellas superior; 215 habitaciones y 12 suites, así como de numerosos recursos, el **Hotel Meliá Almuñécar** supone la creación de numerosos puestos de trabajo y la capacidad para atraer turistas a la zona.*

Los conjuntos de palabras en letra negrita contienen enlaces. Así, por ejemplo, si se selecciona y se activa Costa Tropical se recupera un documento que describe lo que es la costa tropical, que, a su vez, tiene enlaces a otros documentos. Lo mismo se puede decir de los otros dos enlaces del texto: cadena Sol Meliá (describirá las características de este grupo hotelero) y Hotel Meliá Almuñécar (contendrá una descripción detallada del hotel, muy probablemente con precios, fotos, etc.).

Un puntero a otro documento se denomina hiperenlace y un texto con hiperenlaces se denomina hipertexto. Un hipertexto disponible en la www se denomina página. La primera página que aparece al acceder a una dirección web se denomina página principal (*homepage*). Por lo general la página principal se organiza como un índice para acceder a otras páginas sobre el mismo tema en el propio servidor, y cualquier página puede tener hiperenlaces a cualquier servidor web. En la web no sólo se puede acceder a documentos de tipo hipertexto, también hay documentos de hipermedia que contienen gráficos, imágenes y sonidos. En el ejemplo anterior, al activar *Costa Tropical*, podría aparecer un documento con un hiperenlace a un archivo con los sonidos de las olas del mar y de las gaviotas, otro hiperenlace a un vídeo en el que se podría ver la subasta de pescado en el muelle, etc. Denominaremos genéricamente hiperdocumento a un hipertexto o a un hipermedia. El hecho de ir accediendo ("saltando") de un documento a otro a través de la web se denomina navegar a través de Internet.

A continuación se describe brevemente la forma de direccionar los documentos en la web (Secc. 19.3.1), el protocolo http para transferencia de hiperdocumentos (Sección 19.3.2), y, por último efectuaremos una descripción de los programas que posibilitan el acceso y visualización de páginas web (**navegadores**, Sección 19.3.3). El apartado concluye con unas consideraciones finales.

19.3.1 Direccionamiento en la web (URL)

Obviamente uno de los conceptos fundamentales de la web es la forma de localizar los documentos. Esto se efectúa utilizando el **Localizador Uniforme de Recursos** (**URL**), que puede referenciar cualquier tipo de documento en la red. Un localizador URL se compone de tres partes, separadas con rayas inclinadas:

Protocolo-de-recuperación://computador/ruta-y-nombre-del-archivo

Existen distintos protocolos para recuperación de un documento, entre los que se encuentran HTTP, FTP, Gopher, SMTP y TELNET. Computador es el nombre DNS del computador

(Sección 19.4.xx), y, el último campo, es el nombre completo del archivo que contiene la página buscada, incluyendo su ruta de acceso dentro del computador donde se encuentra.

Con frecuencia el computador concreto donde se encuentra la página principal de un dominio se le denomina ***www.nombre-del-dominio***.

EJEMPLO 19.7

Una dirección de una página web puede ser:

http://www.ugr.es/Informatica

lo que significa:

http	*el protocolo de transferencia usado es el http*
www	*la página forma parte de la World Wide Web (www es un alias del nombre del computador)*
ugr.es	*la página pertenece a la Universidad de Granada*
Informatica	*documento o camino hasta el documento solicitado*

EJEMPLO 19.8

La dirección de la página web del Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Granada es:

http://atc.ugr.es

Obsérvese, que se utiliza el protocolo http. El nombre del computador es atc.ugr.es, y al no indicar el nombre del archivo se accederá a la página principal del departamento citado.

Por otra parte la dirección URL del servidor web de la Universidad de Granada es:

http://www.ugr.es

es decir, el computador donde se encuentra es www.ugr.es (www suele ser un alias del nombre del computador)

Obviamente los hiperenlaces son sencillamente localizadores URL, ya que estos determinan exactamente el archivo y el computador donde se encuentra la página así como el protocolo utilizado

19.3.2 Protocolo HTTP

Uno de los pilares de la www se encuentra en el **HTTP** (**Protocolo de transferencia de hipertextos**), un protocolo que sigue el modelo cliente/servidor usado por lo general entre un navegador de web y un servidor web. Realmente lo que establece HTTP es cómo recuperar hiperdocumentos distribuidos y enlazados a través de la web.

El protocolo establece para el cliente tres posibles paquetes para:

- Solicitar datos del servidor (archivos de hipertextos o de hipermedia), que es el objetivo fundamental.

- Solicitar la recepción de información sobre características del servidor (cabeceras).
- Enviar información al servidor (respuestas de formularios, correos electrónicos, etc.).

El servidor tiene definido un único paquete de respuesta que se limita a enviar la información de cabeceras o el archivo solicitado.

Cuando un usuario, a través de su navegador, hace una petición de acceso a un documento de una dirección dada, el HTTP solicita una conexión TCP con la que transmite al servidor la **petición HTTP**, en la que se incluye, entre otras cosas, el URL, información sobre el cliente, y la orden concreta con sus parámetros. El servidor responde realizando la operación requerida y enviando una **respuesta HTTP**, que contiene información sobre el servidor y de control, y la respuesta propiamente dicha a la solicitud (la cabecera o el hiperachivo). Una vez dada la respuesta, automáticamente se cierra la conexión TCP. Si a continuación seleccionamos un hiperenlace del documento visualizado por el navegador, se vuelve a repetir todo el proceso (nueva conexión TCP/IP, etc.), ahora con el URL asociado al enlace.

19.3.3 Navegadores web

Un **navegador** (*browser,* en inglés) es un programa que permite captar, interpretar y visualizar documentos web.

La captación del documento se realiza con ayuda de uno de los protocolos vistos con anterioridad que permiten la obtención de archivos a través de Internet: HTTP, FTP, Gopher, TELNET, etc.

En un documento de texto se pueden diferenciar dos cuestiones: los caracteres que contiene y el formato y presentación de los mismos (márgenes de la página, tipo y tamaño de letra, sangrías, tabulaciones, etc.). Para especificar los caracteres existen convenciones mundialmente aceptadas como son la codificación ASCII y Unicode (Sección 3.3); sin embargo, el formato y presentación de un documento se crea por muy diferentes procedimientos que dependen notablemente del computador de que se trate. Esto presenta un grave inconveniente cuando se capta un documento a través de la web, ya que es necesario visualizarlo en el computador local con el aspecto y formato correcto, independientemente del procedimiento que se utilizó para darle formato en el servidor que contiene la página. Para solucionar el problema se ha definido el estándar **Lenguaje de marcado de hipertextos** (**HTML**), que contiene sólo caracteres ASCII, entre los que se insertan órdenes de formato (marcas), también en ASCII. Las marcas de formato van embebidas en el texto principal y entre ángulos (< >). Así, por ejemplo, la frase:

Era de **noche** y sin *embargo* llovía.

en lenguaje HTML resulta ser:

Era de noche y sin <i>embargo</i> llovía.

Puede observarse que con la marca se da la orden de escribir en negrita, y la significa final de negrita. Por otra parte <i> es inicio de letra cursiva (o itálica) y </i> es el final de este tipo de letra.

Los documentos web pueden clasificarse en tres tipos: estáticos, dinámicos y activos, según se describe a continuación.

- **Documentos estáticos.** Son documentos con información fija. En este caso el servidor web contiene el documento en lenguaje HTML, que envía al cliente a petición de éste, y es procesado por el intérprete de HTML que genera el documento a ser visualizado,

según las características del equipo que accede a la web. La Figura 19.17 muestra un esquema de este proceso. Con frecuencia las páginas HTML se generan con pequeños programas (*scripts*) escritos en **lenguaje JavaScrip**[2]. Existen versiones mejoradas del HTML original como la **XHTML** (HTML ampliado).

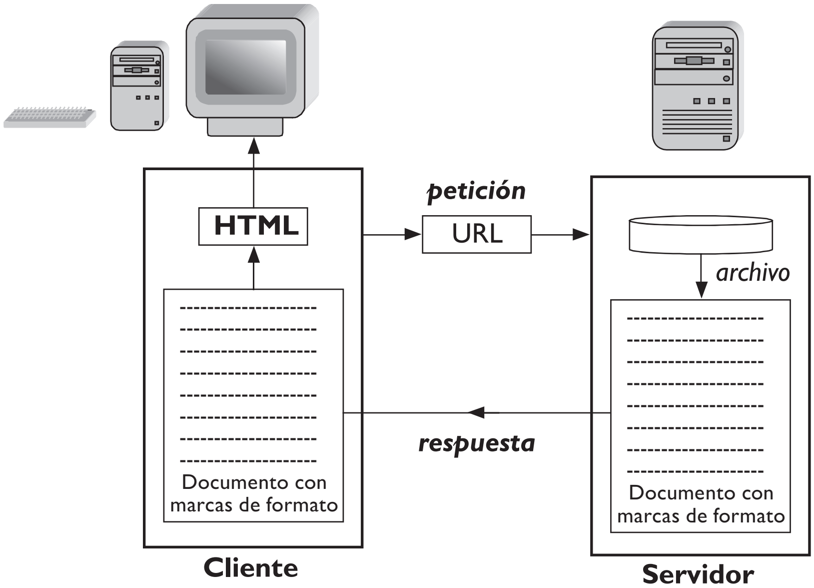


Figura 19.17. *Esquema del funcionamiento del correo electrónico Esquema que muestra los pasos a seguir para visualizar un documento estático.*

- **Documentos dinámicos.** Son documentos cuya información cambia constantemente de un momento a otro. Piénsese, por ejemplo en una página web que presenta los valores de la bolsa en tiempo real, o la hora y temperatura de una determinada localidad. En este caso, cada vez que el usuario solicita una de estas páginas se lanza en el servidor la ejecución de un programa que genera el documento a transmitir o actualiza sus partes dinámicas. Una vez generada la página se transmite al cliente (Figura 19.18). Existe un estándar denominado **CGI** (**Interfaz común de pasarela**, *Common Gateway Interface*) para construir documentos web dinámicos. Los programas CGI generadores de documentos dinámicos pueden ser escritos en lenguajes orientados a objetos, tales como Perl, Python o C++.
- **Documentos activos.** Son documentos que también se generan en el momento de visualizarse, pero en lugar de hacerlo en el servidor se crean en el propio cliente. En este caso el servidor remite el programa y los datos, el cliente ejecuta el programa que genera la página y los movimientos se van actualizando por el propio programa. Los documentos activos se suelen programar en lenguaje Java, un lenguaje orientado a objetos desarrollado por Sun Microsystems que es muy similar a C++. Tiene la particularidad de que los compiladores de Java generan código máquina para un procesador virtual denominado JVM (*Java Virtual Machine*); de forma que para ejecutar este código en una máquina concreta basta con que disponga de un emulador (software) de la JVM. La mayoría de navegadores modernos incluyen un emulador JVM, de esta forma (Figura 19.19) cuando se accede a

[2] No confundir con el lenguaje Java, con el que, salvo el nombre, no tiene nada en común.

un documento activo, el servidor se limita a transmitir el código máquina del programa que genera la página en JVM, y el emulador del cliente la ejecuta, generando la página y controlando sus partes móviles. Un pequeño programa en Java se suelen denominar *applet*, término que puede considerarse como un acrónimo de "pequeña aplicación", en inglés.

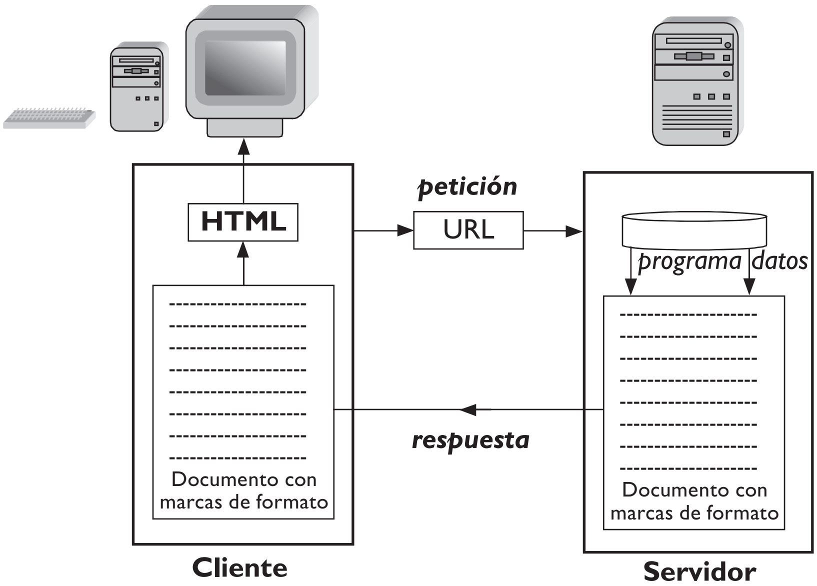


Figura 19.18. *Esquema que muestra los pasos a seguir para visualizar un documento dinámico.*

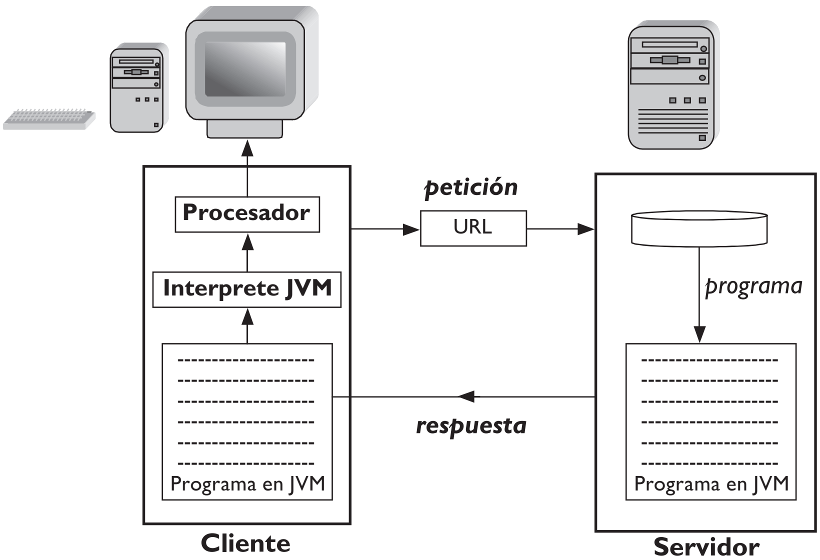


Figura 19.19. *Esquema que muestra los pasos a seguir para visualizar un documento activo.*

19.3.4 Consideraciones finales

La información web está muy dispersa y apenas organizada jerárquicamente. Para facilitar el acceso a documentos desconocidos que traten de temas concretos se han desarrollado **motores** o **robots de búsqueda** (*search engines*). Con ellos los usuarios de Internet pueden localizar archivos a lo largo de grandes bases de datos que contienen enlaces a documentos web. Con estos programas se pueden combinar palabras utilizando los operadores básicos de la teoría de conjuntos (Álgebra de Boole): unión, intersección, negación, y de esta forma canalizar la búsqueda. Entre los motores de búsqueda más conocidos se encuentran los que se reseñan en la Tabla 19.8.

Tabla 19.8. *Algunos sistemas de búsqueda en la web.*

Google	http://www.google.es
MSN Research	http://www.msn.es
Yahoo	http://www.yahoo.com
AltaVista	http://www.altavista.com
Infoseek	http://infoseek.go.com
Webcrawler	http://webcrawler.go.com
Lycos	http://lycos.es
Terra (español)	http://terra.es

Internet en general, y la www en particular, no sólo está sirviendo para obtener información como si se tratase de consultar una biblioteca, sino que está dando lugar a nuevas posibilidades como las siguientes:

- Intercambios comerciales a través de internet (comercio electrónico).
- Telefonía a través de Internet.
- Videoconferencia.
- Difusión de radio y televisión (en vivo).
- Audio bajo demanda.
- Vídeo bajo demanda (películas, documentales, etc.).
- Etc.

La web puede considerarse un medio de difusión completamente revolucionario. Cualquier empresa o persona particular, con muy pequeños recursos, puede crearse su propia página web que será accesible en cualquier parte del mundo. No obstante, existen graves problemas en cuanto a que la información que se incluye en la web no pasa por ningún control o contraste de calidad o ético. Por otra parte, existen muchos documentos incompletos (en construcción) y además la información es muy volátil ya que las referencias no son estables (cambian de lugar) y los documentos pueden ser eliminados en cualquier momento por sus creadores, o, sencillamente el computador que las contiene se da de baja. En contrate con la web, por ejemplo, un artículo científico publicado en una revista de prestigio es revisado y contrastado por un equipo de especialistas y la referencia de su publicación (titulo de la revista, año, mes y páginas) permanece en el tiempo, pudiendo ser localizado en cualquier momento.

19.4 Conclusión

En este capítulo se han analizado aspectos clave sobre la interconexión de computadores para formar redes que permitan el intercambio de información entre ellos y la distribución de recursos.

En primer lugar (Sección 19.1.1) hemos descrito algunos conceptos básicos sobre normalizaciones, tema de suma importancia ya que sin su existencia sería impensable la interconexión de sistemas de la extraordinaria diversidad, en cuanto a su naturaleza y características, de los equipos conectados a las redes de computadores. A continuación, se han presentado distintas topologías para la interconexión de computadores en red (Sección (19.1.2) y el modelo de referencia OSI para interconexión de sistemas de distintas características (sistemas abiertos), Sección 19.1.3. Después, en las secciones 19.1.4 a 19.1.6 abordamos el estudio de diversos tipos de redes: de área personal (PAN), de área local (LAN) y de área amplia (WAN), mostrando las peculiaridades y características más notables de cada una de ellas.

Debido a la importancia que ha adquirido la red de redes (Internet) se ha dedicado a ella la segunda parte de este capítulo. Las redes actuales, y en particular Internet, están formadas por segmentos de red y subredes más o menos complejas. Todos estos sistemas se interconectan por medio de los dispositivos o procesadores de red que se han analizado en la Sección 19.2.1. Uno de los fundamentos de Internet radica en el direccionamiento utilizado para identificar cada una de las estaciones conectadas a ella; a este tema se dedica la Sección 19.3.1. Otras cuestiones básicas son los protocolos que determinan el funcionamiento de Internet (Sección 19.3.2) y el ámbito de aplicaciones (Sección 19.2.4).

La aplicación de Internet que ha suscitado un mayor interés popular es la Web, a la que dedicamos la Sección 19.3 de este capítulo. Analizamos cuestiones tales como direccionamiento en la Web (Sección 19.3.1), el protocolo http (Sección 19.3.2) y los navegadores que posibilitan el acceso eficiente y cómodo a páginas web (Sección 19.3.3). Concluimos la sección con unas consideraciones finales.

Conviene hacer hincapié que este capítulo, al igual que los otros de este texto, únicamente pretende presentar una introducción al tema de redes de computadores; siendo obviamente imposible abordar todas las cuestiones que se plantean dentro de este campo. Una de ellas, y que cada vez está adquiriendo mayor relevancia, es la de la seguridad. Invitamos al lector interesado en este y otros temas relacionados con redes de computadores a consultar los libros indicados en la bibliografía, la mayoría de los cuales son buenos textos dedicados en exclusiva al tema mencionado.

Después de la lectura de este capítulo podemos concluir que existe una importancia creciente en la interconexión de computadores frente a cualquier otro dispositivo de cómputo, siendo un factor esencial la velocidad de comunicación ya que cada vez más las aplicaciones y la información se encuentran distribuidas en la red en lugar de en los computadores locales.

EJERCICIOS

19.1 Explicar las ventajas que tiene realizar una compresión de información antes de transmitir un mensaje. ¿Por qué se realiza en la capa de presentación y no en una inferior?

19.2 Indicar las capas que según el modelo OSI deberían encargarse de las siguientes tareas:

a) Envío y recepción de una transacción o mensaje.
b) Cambios de códigos.
c) Comprimir/descomprimir la información.
d) Establecer y finalizar una sesión.
e) Formar paquetes a partir de segmentos, y formar segmentos a partir de paquetes.
f) Realizar el encaminamiento de los distintos paquetes a través de los dispositivos de la red.
g) Transmitir sin error cadenas de bits en forma de señales ópticas, eléctricas, etc

19.3 Suponga que un protocolo de una red establece unas tramas en las que, por término

medio, un 60% de las mismas contienen los datos del usuario, y el resto es información de control (cabeceras, información para detección de errores, etc.). La velocidad de transmisión de datos (del usuario) que se obtiene en la red anterior es de 500 Mbps.

a) Suponiendo que se modificase el protocolo de forma tal que el porcentaje de datos del usuario se incrementase hasta el 90%, obtener la nueva velocidad de transmisión de datos obtenible.
b) Dibuje una gráfica que muestre la dependencia de la velocidad de transmisión de los datos con el porcentaje de datos útiles en las tramas.
c) ¿Qué modo de transmisión de redes públicas se fundamenta, entre otros conceptos, en la reducción de la sobrecarga de información de control para ofrecer a los usuarios mayores velocidades de transmisión?

19.4 Dada la porción de red mostrada en la Figura 19.20.

a) Indicar los caminos posibles de interconexión entre los equipos C1 y C3.
b) ¿Qué dispositivos de red, y en que capa del modelo OSI, se decide el camino o ruta de un paquete?

19.5 Dada la porción de red mostrada en la Figura 19.20 indicar los caminos posibles de interconexión entre los equipos C2 y C3.

19.6 Indicar las diferencias entre una tarjeta de red y un módem.

19.7 Suponiendo una red en anillo de 32 computadores, en la que cada computador recibe los paquetes del computador anterior, los analiza, y luego los reexpide al computador siguiente, indicar los problemas de fiabilidad que presenta este tipo de red.

19.8 Reproducir esquemáticamente el intercambio de información entre dos equipos B y D de una red local en anillo (*token ring*) que está constituida por cinco equipos (A, B, C, D, E). Suponer la transmisión de varias tramas (algunas de ellas llegando al receptor incorrectamente).

19.9 En un domicilio particular al que llega cable-coaxial de una compañía que ofrece servicios de Internet se ha instalado un cable-módem, al que se ha conectado un punto de acceso que actúa de encaminador inalámbrico. De esta forma se dispone de una red con 4 equipos PC y una impresora, todos ellos con conexión de red inalámbrica. Indicar, de forma razonada,

a) la topología de la red y
b) el tipo de red (LAN, WAN, o MAN).

19.10 ¿Cuántos bits deberían tener las direcciones IP para poder disponer de un equipo conectado a Internet por cada metro cuadrado del globo terráqueo? *Nota*: considerar que el metro, aproximadamente, es la diez millonésima parte del cuadrante de un meridiano terrestre.

19.11 Comparar las ventajas e inconvenientes de los protocolos TCP y UDP.

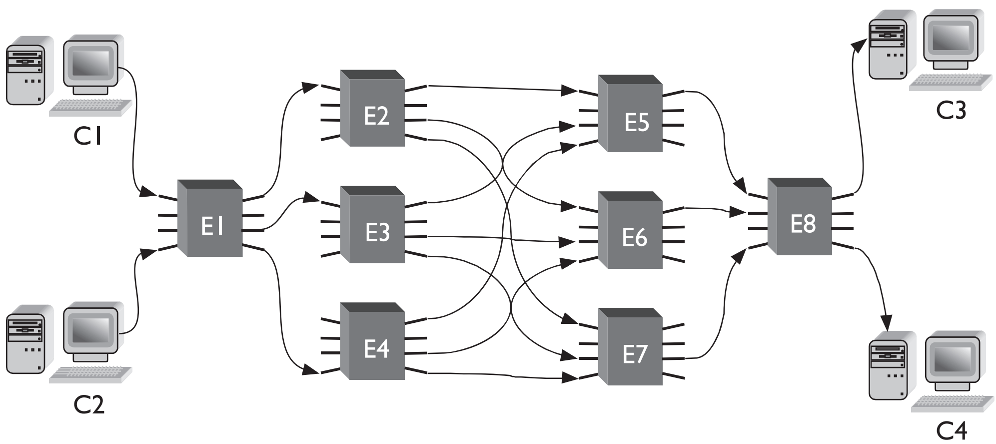


Figura 19.20. *Ejemplo de interconexión de equipos a través de encaminadotes.*

19.12 La InterNIC es una organización que coordinada y asigna conjuntos de direcciones IP. Las asignaciones pueden ser de tres tipos:

- **Clase A:** El primero de los cuatro bytes empieza siempre por 0 y los 7 bits restantes identifican la red (dominio). Los restantes bits, hasta 32, especifican una dirección de computador dentro de cada una de las redes.
- **Clase B:** todas las direcciones empiezan por 10, y el resto del primer byte (6 bits) junto con el segundo byte especifica la red. Los restantes bits, hasta 32, especifican una dirección de computador (estación) dentro de cada una de las redes.
- **Clase C:** todas las direcciones empiezan por 110 y los bits restantes del primer byte (5 bits) junto con el segundo y tercero identifican la red. El último byte especifica un computador dentro de cada una de las redes.

Dada la dirección IP

7B.4A.54.37

Indicar:

a) La clase a la que pertenece la dirección.
b) El dominio al que pertenece esa dirección.
c) El código del computador dentro de su dominio.

19.13 Se dispone de una LAN de tipo Ethernet, bus único, en la que están conectados 64 equipos. Indicar de forma razonada de qué dependerá la frecuencia de que se den colisiones. ¿Cómo se podría reducir el número de colisiones?

19.14 Indicar, justificándolo, cuál es la topología (estrella, bus único o en anillo) de una red LAN más robusta frente a una avería de una parte de la red.

19.15 Hacer una relación de las funciones que debe realizar un:

a) Repetidor.
b) Concentrador.
c) Puente.
d) Conmutador.
e) Enrutador.
f) Pasarela.

19.16 Se observa que el porcentaje de tráfico (paquetes intercambiados) entre los 8 equipos de en una red de área local es el que se indica en la tabla siguiente:

	E0	**E1**	**E2**	**E3**	**E4**	**E5**	**E6**	**E7**	**TOTAL**
E0	–								
E1	3	–							3
E2	2	4	–						6
E3	3	1	4	–					8
E4	9	6	5	4	–				24
E5	2	2	1	5	7	–			17
E6	1	1	2	9	9	2	–		24
E7	0	1	2	2	7	3	3		18
TOTAL	20	15	14	20	23	5	3	–	100

Para mejorar el sistema se decide dividirla en dos subredes e interconectarlas con un puente. Obtener:

a) Una tabla con el porcentaje de tráfico en que está involucrado cada equipo.
b) Los equipos que deben interconectarse a la Subred A y a la Subred B para que ambas subredes estén equilibradas.
c) El porcentaje del tráfico total que tendrá que dejar pasar el puente de la Subred A a la Subred B, y viceversa.
d) La disminución de tráfico de cada subred, con respecto a la red inicial.

19.17 Indicar las diferencias y similitudes entre el modelo OSI y el conjunto de protocolos TCP/IP

19.18 Teniendo en cuenta la asignación de direcciones IP que hace la InterNIC (Problema 19.12); obtener:

a) El número de redes direccionables dentro de de cada clase.
b) El número total de computadores direccionables dentro de una red de cada una de las clases
c) El número total de redes y de computadores direccionables por Internet.

19.19 Teniendo en cuenta la asignación de direcciones IP que hace la InterNIC (Problema 19.12), dada la dirección IP:

138.205.154.8

Indicar:

a) La clase a la que pertenece

b) El dominio al que pertenece esa dirección.
c) El código del computador dentro de su dominio.

19.20 Obtener la dirección en notación de punto decimal de la dirección IP binaria:
3C7F C352

19.21 Obtener la dirección IP binaria de la siguiente dirección IP en notación de punto decimal:
123.241.122.15

19.22 Indicar las diferencias y similitudes entre una dirección IP, una dirección de correo electrónico y una dirección URL.

19.23 Dada la de la siguiente dirección URL:
http://www.rne.es/rc/index.htm
Indicar:
a) El significado de los elementos que contiene.
b) El tipo de documento que es (estático, dinámico o activo).
c) La institución a la que pertenece.

19.24 Dada la de la siguiente dirección URL:
http://www.la-moncloa.es/web/pg06b.htm
Indicar:
a) El significado de los elementos que contiene.
b) El tipo de documento que es (estático, dinámico o activo).
c) La institución a la que pertenece.

19.25 A qué capas y en qué modelo se utilizan las siguientes unidades de datos de protocolo (PDU):
a) Transacción.
b) Mensaje.
c) Paquete.
d) Segmento.
e) Datagrama de usuario.
f) Datagrama IP.
g) Phit.
h) Trama.

19.26 Indicar el significado de las siglas y para qué sirven los siguientes protocolos:
a) HTTP
b) FTP
c) Gopher
d) SMTP
e) DNS
f) RTP
g) UDP
h) TCP
i) IP
j) TELNET

BIBLIOGRAFÍA

[Car02] Carlson, A. B.; Crilly, P. B.; Rutledge, J. C.: *Communication Systems,* 4ª Edc. McGraw-Hill, 2002.

[For01] Forouzan, B.: *Transmisión de datos y redes de comunicaciones*, 2ª Edc. McGraw-Hill, 2001.

[For00] Forouzan, B.: *TCP/IP. Protocol Suite*, McGraw-Hill, 2000.

[]For03] Forouzan, B.: *Local Area Networks*, McGraw-Hill, 2003.

[Gar03] García Teodoro, P.; Díaz Verdejo, J.; López Soler, J. M.: *Transmisión de datos y redes de computadores*, Prentice Hall, 2003.

[Hay94] Haykin, S.: *Communication Systems*, John Wiley & Sons, 1994.

[Kur01] Kurose, J. F.; Ross, K. W.: *Computer Networking,* Addison Wesley, 2001.

[Leo01] León-García, A.; Widjaja, I.: *Redes de comunicación. Conceptos fundamentales y arquitecturas básicas,* MacGraw-Hill, 2001.

[Pri05] Prieto, A.; Prieto, B.: *Conceptos de Informática,* Serie Shaum, McGraw-Hill, 2005.

[Sta04] Stallings, W.: *Comunicaciones y redes de computadores*, 7ª Edc. Prentice Hall, 2004.

[Tan03] Tanenbaum, A. S.: *Redes de computadoras,* 4ª Edc. Prentice Hall, 2003.

[Whi04] White, C. M.: *Data communications & Computer Networks*, Thomson, 2004.

CAPÍTULO 20

Evolución histórica y desarrollo de la informática

"Who controls the past commands the future. Who commands the future conquers the past".
George Orwell

En este capítulo se presenta un resumen de los hitos más importantes que han contribuido al desarrollo de la informática. Una gran parte de los conceptos que se exponen ya han sido analizados a lo largo del texto, por lo que este capítulo tiene una doble pretensión: hacer un pequeño resumen o repaso de los capítulos precedentes e incardinar dichos conceptos dentro de la perspectiva histórica de la informática, lo que reforzará la comprensión de los mismos. También se analizan algunas de las tendencias significativas de la informática en el momento actual. Desde una perspectiva científica, para facilitar el avance de cualquier disciplina de la ciencia o de la ingeniería, es necesario conocer adecuadamente no solamente el presente sino también el pasado, ya que en cierta medida el futuro es una extrapolación de éstos últimos, de aquí que tenga pleno sentido la inclusión del presente capitulo y la cita de George Orwell con que éste se inicia.

20.1 Antecedentes de la informática

Desde la antigüedad, y la mayoría de las veces de forma inconsciente, el hombre ha tratado de reducir al máximo su trabajo. De esta forma surgieron las *herramientas* y las *máquinas*. Con ayuda de estos dispositivos el hombre disminuye la necesidad de esfuerzo físico y se dedica a funciones superiores, como son la puesta en marcha y control de las operaciones a realizar por los instrumentos citados.

Paralelamente al desarrollo de las herramientas y las máquinas, el ingenio humano se dedicó a sustituir también la *inteligencia* aplicada a tareas rutinarias. La **automática** es precisamente la ciencia que trata de la sustitución del operador humano por un operador artificial en la ejecución de una tarea física o mental previamente programada. Aplicada a los procesos industriales,

su fin es suprimir el eslabón *hombre* en la cadena de producción. Se pretende llegar a la fábrica *automática* en la cual las máquinas regulan su propio comportamiento, desde el principio hasta el fin de la producción.

La Informática puede considerarse como la ciencia y tecnología aplicada a la automatización del razonamiento y del tratamiento de la información. Recuérdese (Sección 1.1) que la palabra Informática procede de la yuxtaposición de los vocablos INFORmación y autoMÁTICA. Puede decirse que las raíces de la informática se encuentran en 1º) el desarrollo de métodos, herramientas y máquinas para facilitar la realización de cálculos de forma eficiente y precisa y, 2º) la sistematización del razonamiento, como paso previo a su automatización y a la búsqueda de modelos formales de cálculo.

20.1.1 Desarrollo de las formas de representación de los números y de los métodos de cálculo

Tal vez el concepto más primitivo relacionado con la informática es el de **número**, y relacionado con él la habilidad de contar. Para los hombres primitivos era relativamente fácil distinguir entre 3 árboles y 4 árboles, pero mucho más difícil resulta relacionar 5 árboles con 5 pájaros; es decir, separar los números de las cosas concretas que se cuentan, o, en otras palabras, abstraer el concepto de número. Prueba de esta dificultad es la existencia de culturas que utilizaban distintos lenguajes numéricos dependiendo de los objetos que representaban. Así en el lenguaje Thimshian de los indios de Columbia existían siete conjuntos distintos para designar las cifras numéricas, según se tratase de representar objetos planos y animales, objetos redondos y el tiempo, objetos largos y árboles, canoas, personas, etc. [Wil85].

Se tiene conocimiento de que el hombre, a través de distintas culturas, ha utilizado distintas bases de numeración, entre las que se encuentran las bases 4, 5, 10, 13, 18 y 20. Una dificultad que se presentaba en los pueblos primitivos era contar o representar grandes números. Así en diversas culturas (Tasmana y nativos de Sudáfrica, por ejemplo) se contaba de la siguiente forma: *uno, dos y muchos,* o, *uno, dos, dos y uno,dos y dos, dos y dos y uno,* etc. Obsérvese que esta última forma de contar supone utilizar el sistema de numeración base 3 (véase Capítulo 3). La mayoría de las culturas primitivas utilizaban las bases 5, 10 ó 20, ya que se ayudaban en los cálculos de los dedos, como siguen haciéndolo los niños. Los esquimales utilizaban la base 20, ya que decían: 7 (dedos), ó 1 hombre y 3 dedos (para representar 23), ó 2 hombres y 15 dedos (equivalente a 55).

Los pueblos indo-europeos también utilizaron el sistema base 20 [10]. Una reminiscencia de este hecho es la forma de escribir ciertos números franceses: 80 es *quatre-vingt* (cuatro-veintes), 91 es *quatre-vent-once* (cuatro-veintes-once), etc.

Los sistemas de numeración, desde un punto de vista histórico, pueden clasificarse en sistemas aditivos y sistemas posicionales. Entre los primeros caben destacar, por su difusión, el egipcio y el romano. El sistema indo-arábigo (utilizado en la actualidad) es de tipo posicional (véase Capítulo 4).

Los primeros documentos egipcios referentes a su notación numérica y métodos aritméticos datan del año 3000 a.C., y eran de tipo jeroglífico. El 1 era representado por un trazo vertical, el diez por una horquilla vertical con los dos extremos en la parte inferior, el 100 por una cuerda enrollada, el 1.000 por una flor de loto, 10.000 un dedo apuntando al cielo, el 100.000 un pajarito y un millón por un hombre de rodillas con los brazos en alto en posición de adoración hacia el cielo.

El sistema primitivo romano, utilizaba 7 símbolos: M (1.000), D (500), C (100), L (50), X (10), V (5) y I (1), y con ellos podía expresarse cualquier número menor de 5.000 con una secuencia de símbolos, sin repetir cada uno de ellos más de cuatro veces. En aquella época no se utilizaban las formas reducidas IV = IIII (por 4) y IX = VIIII (por 9). El número 3.524 se representaba así: MMMDXXIIII, y el orden de los símbolos no importaba, aunque se solían ordenar de mayor a menor. La operación de suma es inmediata (de ahí el nombre de *sistemas aditivos*), ya que consiste en mezclar los símbolos de los sumandos, ordenarlos de mayor a menor y simplificar (IIIII = V, etc.). El producto resulta también muy sencillo, ya que se efectúa sumando los números que se obtienen haciendo el producto del multiplicando por cada símbolo individual del multiplicador y, dado el juego de símbolos, sólo están involucradas multiplicaciones por múltiplos de 5 y 10.

EJEMPLO 20.1

Multiplicar CCLXXVII (277) por XXI (21):

	I vez CCLXXVII	*CCLXXVII*
	X veces CCLXXVII	*MMDCCLXX*
	X veces CCLXXVII	*MMDCCLXX*
		MMMMDDCCCCCCLLLXXXXXXVII
y simplificando:		*MMMMMDCCCXVII (= 5.817)*

Por el siglo I ó II, los hindúes (en el Este de Indochina) dieron, en lo que respecta a los sistemas de numeración, tres pasos trascendentales consistentes en:

- La representación posicional de las cifras.
- La base decimal.
- El concepto de cero como un dígito más. Algunos científicos consideran que la invención del cero fue una de los grandes logros de la humanidad. Se tiene constancia de que ya en el año 2000 a.C. los Babilonios utilizaban un símbolo (predecesor del cero) para las posiciones vacías dentro del número, pero les era inconcebible utilizar el cero aisladamente en las posiciones finales (por ejemplo, para representar las dos últimas posiciones de 100).

Esta conjunción de ideas dio al mundo una notación flexible y eficiente. Con ella todo número, por grande que sea, puede representarse por una secuencia ordenada de símbolos tomados de un conjunto de diez.

Las dinastías de los califas instaladas en Bagdad se caracterizaron por absorber las culturas vecinas y de los países que conquistaban, y de esta forma se cree que el sistema de numeración hindú pasó a Bagdad, perfeccionándose allí los distintos procedimientos de cálculo (los términos **aritmética** y **álgebra** son de origen árabe). También se tiene constancia de que Al-Khowârizmî, alrededor del 825, tuvo contacto con astrónomos hindúes, y escribió diversos libros de texto sobre aritmética, álgebra y tablas para cálculos astronómicos para uso en Bagdad. Este libro se difundió mucho por el imperio árabe. Al-Khowârizmî es el creador del concepto de **algoritmo** (este vocablo se deriva del nombre del matemático), que introdujo al desarrollar reglas basadas en operaciones muy elementales para realizar las operaciones aritméticas básicas.

Las primeras constancias escritas de la introducción del sistema indo-arábigo en Europa se encuentran en España en el texto Albelda Cloister (976), y las cifras numerales también aparecen en otro escrito español de 992. Pero realmente quien difundió el sistema de numeración en occidente fue el gran matemático Leonardo de Pisa, más conocido por el nombre de Fibonacci (hacia el año 1200), quien hizo aportaciones de gran interés en la búsqueda de algoritmos para la resolución de problemas concretos.

Un paso de gran relevancia en los métodos de cálculo lo dio hacia 1580 el francés Françoise Viète, que propuso la utilización de letras para representar números desconocidos, lo que supuso un paso trascendental en el desarrollo del álgebra.

Históricamente la siguiente invención de gran importancia se debe a John Napier, que desarrolló la teoría de logaritmos, como ayuda al cálculo, ya que con ellos la multiplicación se podía transformar en sumas, la potenciación en productos, etc.

20.1.2 Sistematización del razonamiento

Paralelamente al desarrollo de los sistemas de representación de los números y de los métodos de cálculo, se fueron ideando formas de sistematización del razonamiento. Los griegos (siglo IV a.C.) sentaron las bases de la **lógica formal**. Ellos desarrollaron la geometría como un sistema de deducción lógica (partiendo de axiomas, postulados, teoremas, etc.), y describieron una metodología para la realización de razonamientos correctos (silogismos, etc.).

Los babilonios y egipcios utilizaron otra metodología distinta a la lógica, la metodología experimental, con la que a base de procesos iterativos de prueba-error obtuvieron tablas de multiplicar, tablas de raíces, tablas de potencias, tablas contables (interés compuesto, por ejemplo), etc.

El siguiente avance digno de destacar en este resumen histórico se debe a Leibniz (1646-1716), que sentó las bases de la **lógica simbólica**, y sugirió la utilización de la aritmética binaria para hacer automáticamente razonamientos. Leibniz dedicó grandes esfuerzos a desarrollar algoritmos y a tratar de abstraer la esencia de los mismos, buscando un método general que permitiese resolver cualquier problema.

Georges Boole (1815-1869), profesor de filosofía en un Colegio inglés, publicó dos libros de una gran importancia, en 1847 el libro *Análisis matemático de la lógica, como ensayo hacía un cálculo de razonamiento deductivo* y en 1854 *Investigación sobre las leyes del pensamiento*. A partir de profundizar en el pensamiento de Aristóteles, y de sus amplios conocimientos matemáticos, asociaba a funciones lógicas (*unión, intersección, complementación*, etc.) y a variables lógicas (*verdadero/falso*), símbolos matemáticos (+, A, –, 1, 0). Desarrolló un álgebra *(Algebra de Boole)* aplicable a este tipo de funciones y variables. Sus trabajos fueron fundamentales para el desarrollo de la lógica matemática (en cierta medida un álgebra para la lógica) y de procedimientos sistemáticos para el diseño de los circuitos que componen un computador (véase Capítulo 4). En su momento fueron totalmente ignoradas las ideas de Boole, pues tanto a los filósofos como a los matemáticos les parecieron una extravagancia. Fue Shannon, en 1938, quien propuso utilizar el álgebra de Boole para analizar las redes de conmutadores (Sección 20.1.4).

El matemático David Hilbert (1862-1943), como otros muchos matemáticos de su época, trató de aplicar a distintos campos de las matemáticas el razonamiento axiomático, e ideó el **cálculo de proposiciones**. Su proyecto concreto iba más lejos, ya que pretendía encontrar un sistema axiomático lógico-matemático, del cual podrían deducirse todas las matemáticas. Concretamente pretendía encontrar un método para establecer si una proposición matemática era falsa o verdadera. Kurt Gödel demostró (1931) que la pretensión de Hilbert era irrealizable ya

que probó la existencia de proposiciones formalmente indecidibles dentro de sistemas axiomáticos consistentes (**Teorema de incomplitud**).

En 1936 el británico Alan Turing publicó *On Computable Numbers*, en el que describió una máquina teórica (**Máquina de Turing**) que podía particularizarse para realizar cualquier cálculo realizable por cualquier máquina real. También en esa década otros investigadores relevantes (como Emile L. Post, Alonzo Church y S. C. Kleene) hicieron brillantes aportaciones en la búsqueda de modelos formales de cálculo. Estos trabajos son hoy día considerados como el origen de la Informática Teórica.

20.1.3 La era mecánica

El hombre primitivo usó los diez dedos de las manos para contar (de ahí nace el sistema de numeración *decimal*), posteriormente se ayudó de guijarros o piedras agrupadas en montones y se sofisticó el método de contar haciendo surcos en una bandeja cubierta de arena (**tablas de arena**), en la que se representaban los números por piedras y sus posiciones en los surcos. En las tablas de arena, el primer surco de la derecha puede corresponder a las unidades, el segundo a las decenas, el tercero a la centenas, y así sucesivamente. La suma consiste en añadir piedras en la hendidura derecha; cuando se completa con diez, se quitan todas y se añade una en la fila siguiente, y así sucesivamente. La palabra *cálculo* procede de *calculus* que significa piedra.

Históricamente puede considerarse que la primera herramienta eficaz para ayuda del cálculo fue el ábaco (Figura 20.1). La tabla de arena puede considerarse precursora del ábaco, que en una primera forma evolucionada, era una tabla de madera con hendiduras en forma de surcos en los que se introducían las *cuentas* (piezas con las que se hacían las cuentas). No está claro el origen del ábaco, algunos autores afirman que se desarrolló por el año 3000 a.C. en el valle entre el Tigris y Éufrates; ahora bien, es muy probable que fuese inventado independientemente por varias culturas, como lo prueba el hecho de que en el descubrimiento de América se encontró que varias civilizaciones precolombinas de Méjico y Perú lo utilizaban. Se tiene noticia, por los escritos de varios autores (Plinio, Juvenal y Cicerón), que los romanos utilizaron varios tipos de ábacos. Algunos disponían de ranuras adicionales para facilitar las operaciones con fracciones. Posteriormente los rusos insertaron las piedras o cuentas en alambres. El ábaco, en su presente forma, fue introducida en China sobre el 1200 d.C., pasando desde allí a Corea hacía el año 1400, y a Japón sobre 1600.

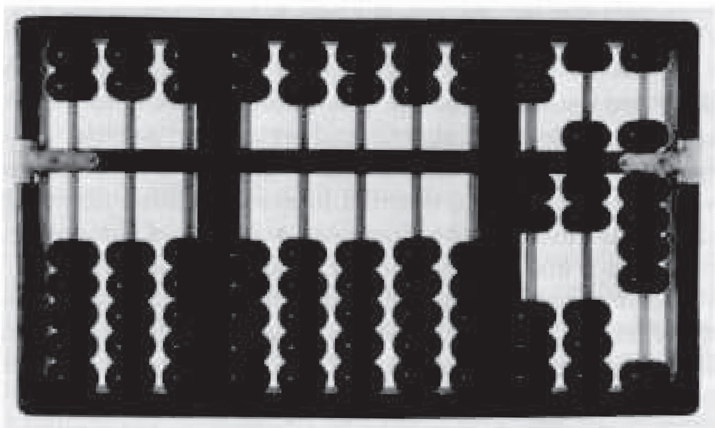


Figura 20.1. *Un ábaco chino conteniendo el número 110.345.678.900.*

El escocés John Napier (1550-1617), además de inventar los logaritmos, ideó en 1615 un sencillo instrumento mecánico denominado **Varillas de Napier**, con el que se podían realizar con gran facilidad multiplicaciones y divisiones.

Entre 1620 y 1630 varios inventores ingleses (Gunter, Oughtred y Delamain) desarrollaron diversas **reglas de cálculo**. Estos instrumentos utilizan logaritmos para realizar diversas operaciones, y los operandos y resultados se representan por longitudes (son sistemas analógicos). Hay que señalar que hasta mediados de la década de los 60 (antes del desarrollo de las calculadoras de bolsillo), los investigadores e ingenieros hacían sus cálculos con ayuda de este tipo de instrumentos.

En 1642 el francés Blas Pascal inventó una máquina para sumar y restar, que patentó en 1647. La máquina está basada en una serie de ruedas giratorias o diales, a las que se encuentran solidariamente unidas unas ruedas dentadas. Con los diales (que recuerdan a los tradicionales de marcado en teléfonos) se introducen los datos. Al girar el dial gira un tambor que tiene escritas las diez cifras decimales. A través de una pequeña ventana o visor puede leerse el número que indica la posición del tambor, que es la cifra acumulada. Cada posición dentro del número tiene su correspondiente dial y tambor de números. El problema está en el arrastre de valores de una posición a otra de mayor peso (de unidades a decenas, decenas a centenas, etc.). Esto se hace engarzando las ruedas dentadas de cada dos posiciones de pesos consecutivos. Las ruedas dentadas están diseñadas de forma tal que cada vez que una rueda, A, da una vuelta completa (esto es, la cifra correspondiente debe pasar de 9 a 10) hace girar la rueda B, correspondiente a la posición de peso superior, 1/10 de vuelta, avanzando la cifra de su visor una posición, y la rueda A se sitúa en cero, por haber empezado una vuelta.

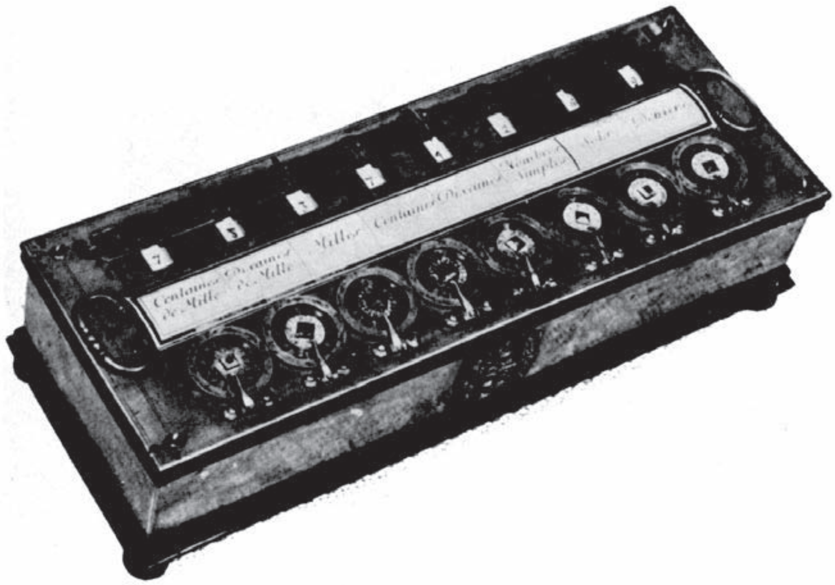


Figura 20.2. *Máquina de calcular de Pascal.*

La siguiente contribución importante se debe al alemán Gottfried Leibniz, del que ya hemos citado su contribución a los modelos de cálculo (Sección 20.1.2). En 1671 construyó una máquina

que realizaba las cuatro operaciones básicas. Su idea consistió en sustituir las ruedas dentadas por cilindros dentados, cuyos dientes varían en longitud. Está máquina operaba correctamente cuando no se presentaban situaciones en las que debían propagarse simultáneamente un número determinado de cifras.

Las máquinas de Pascal y Leibniz sirvieron de base para toda una serie de mejoras que se desarrollaron a lo largo de los siglos XVIII y XIX. Hacia 1820 se comercializó una máquina denominada **Arithmometer** ideada por el francés Thomas de Colmar, y que era de una gran precisión y robustez. En 1885 la compañía Brunsviga comercializó una máquina denominada *Dupla* que fue utilizada (con sucesivas versiones mejoradas) hasta finales de la década de 1950 (Figura 20.3). Ya en 1912 se produjeron 20.000 máquinas de este tipo, que eran utilizadas ampliamente en oficinas y empresas como ayuda en la realización de cálculos rutinarios (contabilidad, etc.).

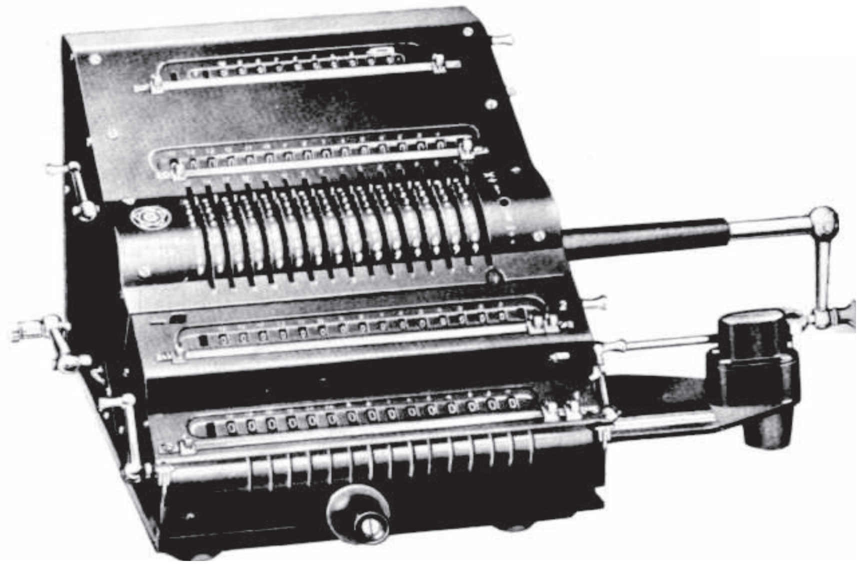


Figura 20.3. *Máquina de calcular de sobremesa Dupla, comercializada por la empresa Brunviga (1885).*

Una innovación muy importante, y en principio ajena a la informática, tuvo lugar a principios del siglo XIX: la introducción de la **tarjeta perforada**. Joseph Jacquard presenta en 1801 en París un telar automático controlado por tarjetas perforadas. Para conseguir un dibujo dado en un telar es necesario levantar determinadas hebras durante cada desplazamiento de la lanzadera. Las hebras que deben moverse juntas van atadas en un extremo a una varilla. El sistema de Jacquard contenía un conjunto de tarjetas, perforadas según el motivo que se desease tejer. Las tarjetas se posicionaban sucesivamente en una estación frente a las varillas. Las varillas que se alinean con una perforación se elevan, y las que no, no se desplazan, con lo que se va controlando la figura que se teje. Sin más que cambiar la secuencia de tarjetas se puede diseñar cualquier dibujo.

El inglés Charles Babbage, en 1822, diseñó una **máquina de diferencias** (Figura 20.4) para producir tablas de navegación. Las máquinas de diferencias se basan en el procedimiento

denominado de diferencias finitas para evaluar polinomios. Se trata de obtener los valores de, por ejemplo, la función $F(x) = x^2 + 3x + 2$, para distintos valores enteros de x. La Tabla 20.1 da los valores de x, F(x), las diferencias de dos valores sucesivos de F(x), y las diferencias de dos diferencias sucesivas. Se observa que las segundas diferencias, d', son constantes. Si el polinomio fuese de orden n, las n-ésimas diferencias serían constantes. La Tabla 20.1 nos sugiere que si conocemos el valor de la segunda diferencia (d'), F(5), y la diferencia primera $d_{54} = F(5) - F(4)$, podemos calcular fácilmente F(6), ya que se verifica:

$$F(6) = F(5) + d_{6,5}$$

$$d_{6,5} = d_{5,4} + d'$$

Tabla 20.1. *Ejemplo de evaluación del polinomio $F(x) = x^2 + 3x + 2$ utilizando el método de las diferencias finitas.*

Fase manual (cálculo de los 3 primeros valores)		
$F(x) = x^2 + 3x + 2$		
F(0) = 0 + 0 + 2 = 2		
F(1) = 1 + 3 + 2 = 6	$d_{1,0} = 4$	
F(2) = 4 + 6 + 2 = 12	$d_{2,1} = 6$	d' = 2
F(3) = 9 + 9 + 2 = 20	$d_{3,2} = 8$	d' = 2

Evaluación por diferencias sucesivas		
x	$\mathbf{d_{i,i-1}}$	$\mathbf{F(i) = F(i-1) + d_{i,i-1}}$
0	–	2
1	4	2 + 4 = 6
2	4 + 2 = 6	6 + 6 = 12
3	6 + 2 = 8	12 + 8 = 20
4	8 + 2 = 10	20 + 10 = 30
...	......	......

Aplicando iterativamente la fórmula anterior podemos hacer una tabla de los valores de cualquier polinomio, sin más que conocer la diferencia constante, d', y la diferencias anteriores. El interés práctico de la evaluación de polinomios radica en que cualquier función (logarítmica, trigonométrica, etc.) puede aproximarse por un polinomio. Una vez hecha esta aproximación, con la máquina de diferencias se podrían obtener tablas de dichas funciones (véase Ejercicio 20.2).

La máquina de diferencias puede considerarse que era un computador digital con un programa fijo. A Babbage, mientras diseñaba la máquina de diferencias, se le ocurrió la idea de que podía modificar las interconexiones entre los registros durante el proceso de cómputo, con lo que obtendría un computador de uso general. De los telares de Jacquard obtuvo la idea de utilizar tarjetas perforadas para controlar las interconexiones y dar la entrada de datos. Los elementos que realizaban las operaciones aritméticas los denominó *taller* (*mill* en inglés), necesitaba otros elementos para almacenar los números, a los que denominó *almacén* (*store*). Puede observarse, que se adelantó a la concepción actual de un computador en cuanto a que proponía cuatro elementos fundamentales: entradas, salidas, unidad aritmético lógica (el taller)

y memoria (el almacén), y en cuanto que su máquina estaba concebida como un computador universal completamente automático y capaz de resolver gran cantidad de problemas; no estaba diseñada para resolver uno concreto. El sistema ideado por Babbage (1833) se denominó **máquina analítica**, y no llegó a construirla, ya que era de una gran complejidad mecánica y se presentaban situaciones en las que se necesitaba el movimiento simultáneo de una gran cantidad de engranajes, y la mecánica de precisión de su época no estaba lo suficientemente desarrollada.

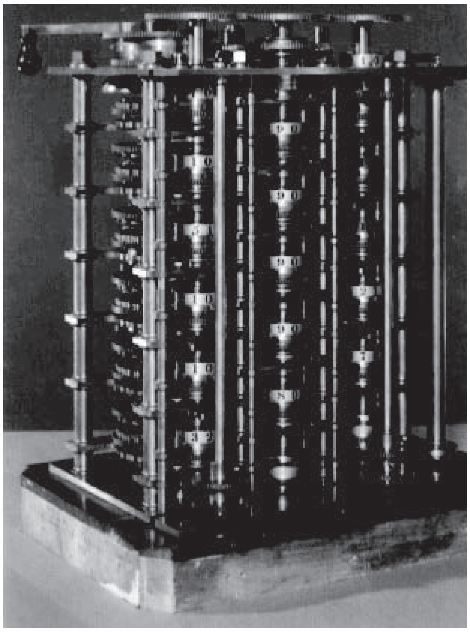


Figura 20.4. *Réplica de la máquina de diferencias de Charles Babbage.*

20.1.4 La etapa electromecánica

Un problema grave que se le presentó al Gobierno de Estados Unidos al final de la década de los años 1880 era que con los medios de que disponía preveían que tardarían en realizar el censo de 1890 12 años, con lo que se solaparía con el de 1890 (en Estados Unidos el censo se hace cada 10 años). Hermann Hollerith, que trabajaba en la oficina de censos, concibió la idea de usar tarjetas perforadas para mecanizar el procesamiento de datos de los censos, desarrollando así una máquina denominada **tabuladora** (Figura20.5). En la tabuladora construida por Hollerith las tarjetas van posicionándose una a una en una estación de lectura, cuyo soporte es una placa conductora. Unas varillas metálicas telescópicas entran en contacto con la superficie superior de la tarjeta; en los sitios que hay perforación las varillas tocan la placa metálica, cerrándose así un contacto eléctrico. Este circuito provoca el desplazamiento de un dial (existe un dial por cada posible posición de perforación), que va contabilizando el número de tarjetas que tienen perforación en la posición correspondiente y hace que la tarjeta vaya al cajetín de clasificación correspondiente. La velocidad de clasificación que obtenía era de unas 60 tarjetas por minuto. Esta máquina tuvo un gran éxito (se expuso en la exposición universal de París de 1889), e hizo posible que el censo de 1890 se efectuase en tres años, en lugar de los doce inicialmente previstos.

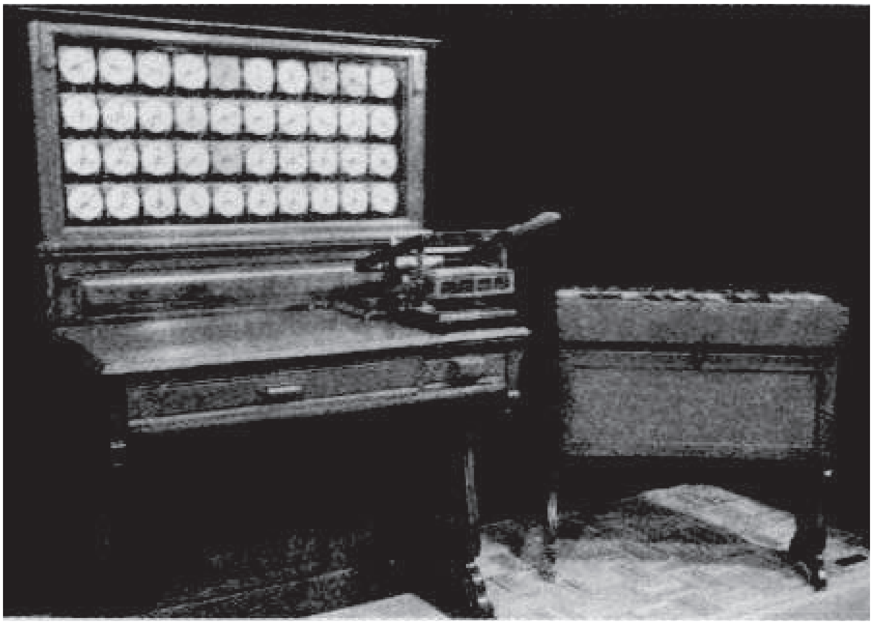


Figura 20.5. *Tabuladora electromecánica para clasificar desarrollada por Hollerith.*

A partir de entonces se incrementa el desarrollo de distintos tipos de calculadoras mecánicas y electromecánicas basadas en las ideas de Pascal, Leibniz y Hollerith. Se crean varias empresas que diseñan y comercializan los nuevos productos: IBM en 1924, Remington-Rand en 1927, Bull (en Francia) en 1931, etc.

Hacia 1914 el español Leonardo Torres Quevedo (1852-1936) creó el primer sistema auténticamente automático para jugar al ajedrez. Posteriormente publicó un trabajo sobre el diseño de una máquina de calcular electromecánica basada en las ideas de Babbage, que previamente había presentado en París en 1920.

De octubre a noviembre de 1939 John Atanasoff, del Iowa State College, junto con su discípulo Clifford Berry, construyó un prototipo de computador digital electrónico que utilizaba aritmética binaria, con el objetivo de automatizar cálculos de física. Aunque nunca concluyó su computador (denominado ABC) que empezó a concebirlo en el invierno de 1937, su desarrollo influyó notablemente en el ENIAC, que será descrito más adelante.

En 1938 Zuse concluyó la construcción de un computador binario electromecánico, el **Z1**. Un año antes, 1937, el profesor Howard Aiken de la Universidad de Harward inició la construcción de un computador denominado **Mark I**, que no concluyó hasta 1944 y que fue el primer calculador digital automático de uso general. Estaba principalmente proyectado para realizar tablas matemáticas y realizaba sumas y restas de 23 dígitos en 3 décimas de segundo, multiplicaciones en seis segundos y divisiones en doce. Los elementos principales de este computador eran relés. Los **relés** son conmutadores (Figura 20.6) que contienen un contacto mecánico que puede abrir o cerrar un circuito eléctrico, aplicando una pequeña corriente en un electroimán (que atrae o repele la lengüeta del contacto). El Mark I utilizaba un gran número de tabuladoras Hollerith, y utilizaba una lectora de cinta de papel, en la que se perforaban las instrucciones, para controlar su funcionamiento.

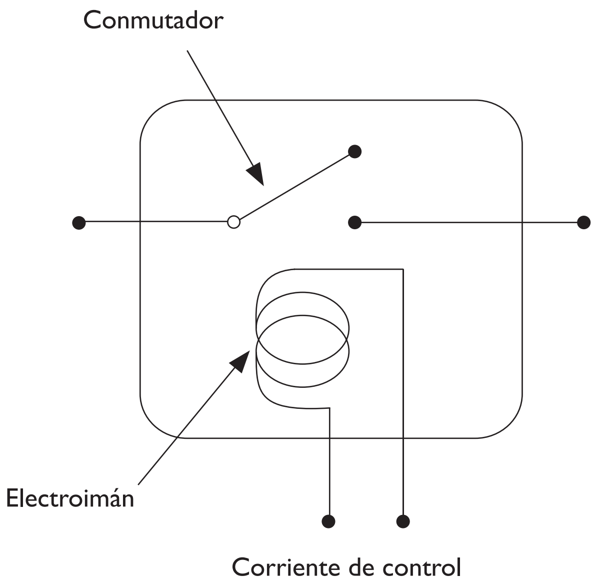


Figura 20.6. *Esquema simplificado de un relé.*

Por esa época, Claude E. Shannon presentó en el Instituto Tecnológico de Massachussetts (MIT) su tesis doctoral y publicó un trabajo (1938) en el que proponía aplicar el Álgebra de Boole binaria para el diseño sistemático de circuitos lógicos complejos utilizando conmutadores. De esta forma con conmutadores (relés, en aquella época) y aplicando los procedimientos ideados por Shannon, se pueden implementar de forma sistemática las funciones aritméticas, lógicas y de control que necesita un computador. Los trabajos de Shannon se referían fundamentalmente a sistemas combinacionales, y D. A. Huffman en 1954 amplió las ideas de Shannon a sistemas secuenciales, considerándolos como máquinas de estados finitos.

20.2 Primera generación de computadores (1946-1954)

Desde el año 1946 a la actualidad la evolución de los computadores se suele agrupar en cuatro generaciones. Lo que más claramente caracteriza cada generación es la tecnología usada, aunque analizaremos también brevemente, para cada una de ellas, la evolución de los sistemas operativos y el software. Conviene indicar que distintos autores consideran intervalos de tiempo para las distintas generaciones, aunque casi todos están de acuerdo con los hechos más notables de cada una de ellas. La evolución de los lenguajes de programación también se agrupa en cuatro generaciones, pero no coincide en los años con las de las generaciones de computadores. No obstante a ello, preferimos seguir un orden cronológico por lo que insertaremos los principales hitos en el campo de los lenguajes de programación dentro de las generaciones hardware. Posteriormente incluimos un apartado (Sección 20.6) aclaratorio sobre las generaciones de los lenguajes que los versados en este campo suelen considerar.

En la Tabla 20.2 se incluye un resumen comparativo de las cuatro generaciones. A continuación se indican las características más relevantes de la primera generación. En las siguientes secciones (Secciones 20.3 a 20.5) analizaremos las restantes.

Tabla 20.2. *Resumen comparativo de las cuatro generaciones de computadores.*

	Primera	Segunda	Tercera	Cuarta
Años	1946-54	1955-63	1964-70	1971-
Tecnología (dispositivos y período de reloj)	Tubos de vacío Memorias de líneas de retardo ms	Transistores Memorias de núcleos de ferritas µs	C.I. (SSI y MSI) Memorias de C.I. ns	C.I. (LSI y VLSI) Memorias DRAM ns
Arquitectura y Estructura	Aritmética de punto fijo	Aritmética exponencial. Registros índices Procesadores de E/S	Microprogramación Memorias cachés Memoria Virtual	Microprocesadores Arquitecturas RISC y paralelas
Sistemas Operativos	sin S.O.	Monitores "batch"	Multiprogramación Multiprocesamiento	S.O. en red S.O. distribuidos
Computadores representativos	ENIAC, EDSAC; IAS; UNIVAC I IBM 702,650	IBM 1401, 1620, 7094 UNIVAC 1004 CDC 6600; PDP-1	IBM 360, 370 DEC PDP-8; UNIVAC 1100	Cray-1, DEC VAX, IBM-*PC* , MIPS, SPARC, RS-6000; HP 9000/750

	Generaciones en la evolución de los lenguajes de programación			
Años	-1952	1949-56	1954-actual	1961-actual
Lenguajes	L. máquina, Subrt."abiertas"	Rutinas interpretativas Ensambladores Programas de ayuda Primeros compiladores	FORTRAN, COBOL, LISP, ALGOL, SNOBOL, Pascal, Ada	Entornos de progr. Paquetes integrados L. declarativos: C++ Prolog, Smalltalk

C.I.: Circuito Integrado; *L:* Lenguaje; *WS:* estaciones de trabajo; *PC:* computadores personales.

20.2.1 Tecnología

La primera generación de computadores (desde 1946 hasta aproximadamente 1957) utilizaba dispositivos electrónicos, y más concretamente tubos de vacío.

Los computadores mecánicos tenían dos serios inconvenientes: la velocidad de cómputo estaba limitada por la inercia de las partes móviles, y la transmisión de información por medios mecánicos (engranajes, ejes, etc.) era engorrosa y nada fiable. Sin embargo, en los computadores electrónicos las partes móviles son los electrones, y la información es transmitida por corrientes eléctricas que se desplazan a velocidades próximas a la de la luz (0,3 metros/nanosegundo).

El **tubo de vacío** más sencillo es el **diodo** o **válvula termoiónica** (Figura20.6). Un diodo está constituido por una ampolla de vidrio, dentro de la cual se ha hecho el vacío, y en cuyo interior existen dos electrodos. Uno de ellos es el cátodo (K), que al ser calentado por una pequeña resistencia interna emite electrones por **efecto termoiónico**. Otro de los electrodos es el ánodo (A), al que si externamente se le aplica una tensión positiva atrae internamente a los electrones producidos por el cátodo; es decir, el triodo conduce internamente la electricidad. Sin embargo, si al ánodo se le aplica una tensión negativa, repele a los electrones, no existiendo conducción. Esta propiedad de conducir la electricidad en un sólo sentido se denomina **rectificación**, y era una propiedad muy buscada en la época ya que con ella se podían detectar ondas electromagnéticas de radio. En 1906 Lee de Forest (1873-1961) patentó el **triodo**, que era un diodo al que se le había introducido un tercer electrodo: la **rejilla** (G), y fue el tubo de vacío de mayor interés

práctico (1906 es considerado por muchos como el año de nacimiento de la Electrónica). La rejilla (G) estaba ubicada entre el ánodo y cátodo, y, aplicando una tensión en ella, se controlaba el paso de electrones entre cátodo y ánodo. Los aspectos más interesantes del triodo es que aplicando una tensión variable entre rejilla y cátodo, se obtenía la misma tensión **amplificada** en el circuito entre ánodo y cátodo (fundamento de los amplificadores electrónicos), y que podía actuar como un conmutador controlado por el circuito de rejilla. Con él, por tanto, aplicando los procedimientos propuestos por Shannon, Huffman y otros, se pueden diseñar los circuitos básicos que conforman un computador (Capítulo 4). En 1919 Eccles y Jordan publican un trabajo en el que presentan un circuito con triodos que actúa como **biestable**, lo que permite utilizar los triodos como circuitos generadores de pulsos y elementos de memoria.

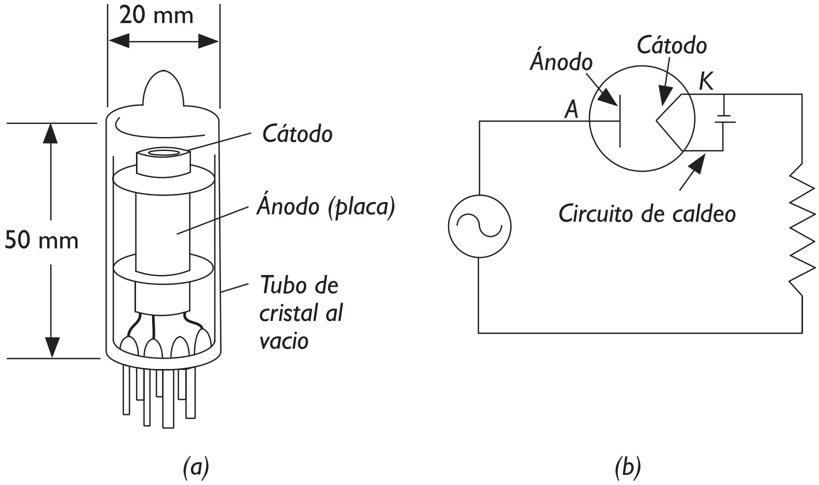


Figura 20.7. *(a) Dibujo simplificado de un tubo de vacío (válvula termoiónica o diodo); (b) Rectificador con diodo de vacío.*

Las memorias construidas con tubos de vacío resultaban extremadamente caras, por lo que se pensó en otra tecnología basada en recircular las señales por unas líneas de retardo de mercurio. La información se transmitía en este medio a velocidades relativamente lentas con lo que recirculándola se podía mantener allí. Una línea de retardo de mercurio almacenaba unos 512 bits, y el coste de almacenamiento del bit resultaba unas 100 veces menor que con tubos de vacío.

En cuanto a elementos de memoria externa se usaron ampliamente las cintas magnéticas. En 1948 Andrew Donald Booth inventó el tambor magnético, que es el dispositivo precursor de los discos magnéticos. Consistía en un cilindro que giraba sobre su eje y cuya superficie estaba recubierta por un material magnetizable similar al de las cintas magnéticas (Sección 10.3.3). Los tambores magnéticos comenzaron a comercializarse en 1952, y su principal inconveniente estaba en que ocupan mucho volumen y consumían gran cantidad de energía.

20.2.2 Computadores

En una reciente desclasificación de documentos británicos de la Segunda Guerra Mundial se ha descubierto que el primer dispositivo de cálculo totalmente electrónico (construido con tubos de vacio) fue la **Colossus**. Esta máquina fue construida por Alan Turing, Tommy Flowers y M. H.

Newman. con objeto de descifrar los mensajes militares secretos del ejército alemán,codificados con la máquina **Enigma**. La Colossus era un computador no programable que se diseñó con un objetivo concreto; es decir, era de uso específico (Sección 1.5).

El primer computador electrónico de uso general que funcionó satisfactoriamente fue la ENAC (Figura 20.8). La construyeron entre el 31 de mayo de 1943 y 1946 el Dr. J. Presper Ecker, físico, y John Mauchly, ingeniero eléctrico, de la Escuela Moore de la Universidad de Pennsylvania. La motivación de la construcción de este computador fue la misma que la de Babbage al construir su máquina de diferencias: realizar automáticamente tablas matemáticas. En este caso eran tablas balísticas para la Armada de los Estados Unidos, que financiaba el proyecto. Contenía cerca de 18.000 válvulas de vacío y unos 1.500 relés. La programación se realizaba por medio de interconexiones en un panel de control. Para cambiar de un programa a otro era necesario extraer el panel de control, cambiar manualmente las interconexiones y volverlo a insertar. Más tarde el control se realizó con tarjetas perforadas. Las entradas y salidas de datos se efectuaban con tarjetas perforadas. Era capaz de efectuar 300 operaciones aritméticas por segundo (en 3 ms realizaba multiplicaciones de 10 dígitos). Su memoria de trabajo contenía 20 acumuladores, cada uno de los cuales memorizaba un número decimal de 10 cifras. Cada cifra decimal se almacenaba en un contador de anillo formado por 10 biestables de triodos; es decir, utilizaba aritmética decimal. El 7, por ejemplo, se representaba poniendo a cero todos los biestables, menos el séptimo que se ponía a 1. El ENIAC pesaba más de 30 toneladas y requería para su instalación más de 140 m^2 de superficie.

Durante muchos años se ha considerado el ENIAC como el primer computador moderno, peo en 1973 un juicio en los Estados Unidos estableció que este honor le cabe a John Atanasoff (Sección 20.1.4).

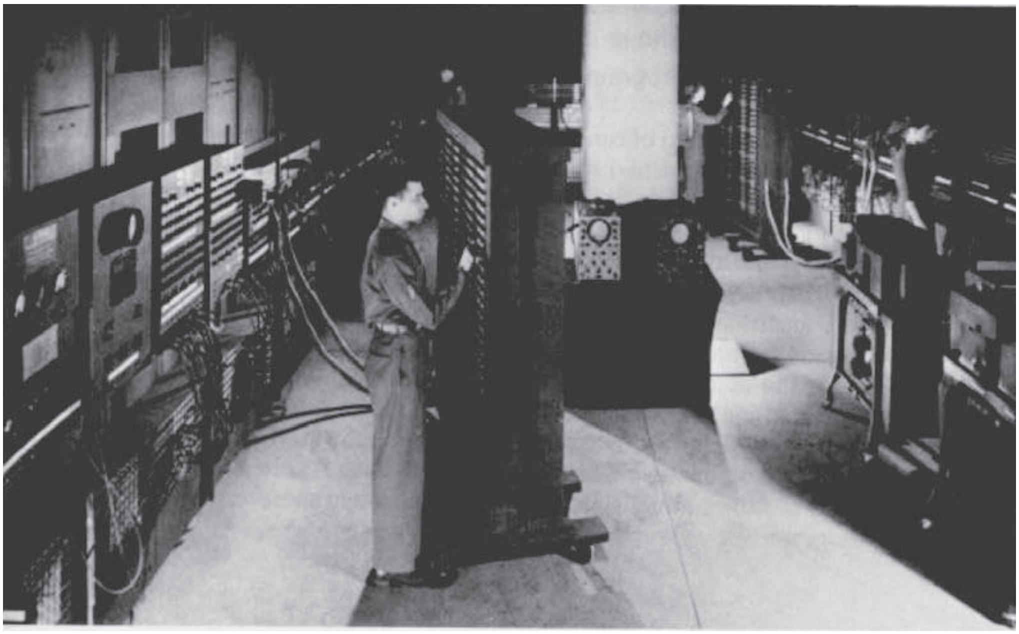


Figura 20.8. *ENIAC, primer computador electrónico de uso general. (Fotografía de H.H.Goldstine. Institute for Advanced Studies, "The Electronic Numerical Integrator and Computer (ENAC)", MTAC, vol.2, nº15, pp. 97-110,1946).*

En el verano de 1944 John von Neumann (1903-1957) comenzó a colaborar en la construcción del ENIAC. Neumann era un científico de gran prestigio, que había cursado las carreras de química, física y matemáticas, y se interesó en la construcción del ENIAC como matemático (colaboraba con Hilbert en su pretensión de axiomatizar las matemáticas) y físico (necesitaba una herramienta potente de cálculo para resolver problemas de dinámica de fluidos). Contribuyó a la concepción de un nuevo computador, el EDVAC (1950), evolucionado del ENIAC, y sus sugerencias (muchas de ellas no personales, ya que surgieron de discusiones con Ecker y Mauchly) más relevantes se centraron en [Ran70]:

- Utilizar la diferenciación realizada por Babbage para las distintas unidades funcionales del computador: Entrada, Salida, Unidad Aritmética, Unidad de Control y Memoria.
- Utilizar una representación binaria para la realización de operaciones aritméticas (el ENIAC, como hemos indicado anteriormente, utilizaba una aritmética decimal).
- El concepto de programa almacenado en la memoria. La memoria debe utilizarse tanto para almacenar datos como instrucciones.
- El ENIAC contenía 20 acumuladores que podían operar simultáneamente, esto implicaba un diseño muy complejo. Neumann sugirió secuenciar todas las operaciones; es decir, descomponer las tareas en secuencias de pasos elementales realizables uno detrás del otro.

La mayoría de computadores actuales siguen estos criterios y por ello se suelen denominar *arquitecturas von Neumann* (véase Sección 1.2.1).

Aunque, como indicamos anteriormente, la idea de programa almacenado se debe a Neumann (la publicó en 1945), fue utilizada por Maurice Wilkes, de la Universidad de Cambridge, para construir entre 1945 y 1949 el **EDSAC** (*Electronic Storage Automatic Calculator*), anticipándose así a la conclusión del **EDVAC** (1952) en cerca de dos años. Wilkes introdujo posteriormente (1953) el concepto de **microprogramación** para el diseño de unidades de control (Sección 7.2.2), aunque no pudo llevarlo a la práctica hasta años después, debido a que no se disponía de una tecnología de memoria suficientemente rápida.

Entre 1947 y 1951 en el MIT se construyó el primer computador para aplicaciones de tiempo real, el **WirlWind**. Su objetivo era procesar señales de radar y simular aeronaves; además fue el primer computador que utilizó memorias de ferrita.

En 1951 se vendió el primer computador comercial fabricado en serie: la UNIVAC (*UNIVersal Automatic Computer*). Su uso era general, de forma que se utilizó tanto para aplicaciones comerciales como científico-técnicas. Como memoria disponía de líneas de retardo de mercurio, y podía realizar 1000 operaciones por segundo. La primera unidad construida fue adquirida por la oficina de censos de Estados Unidos; estuvo en servicio durante doce años constantemente (las veinticuatro horas del día).

En 1952 IBM entró en la comercialización de computadores con la IBM 701, de la que vendieron 19 unidades. Un computador que puede considerarse como fin de esta primera generación fue el IBM 704, tenía registros índices y coma flotante por hardware, utilizaba un *programa de control* que puede considerarse un rudimentario sistema operativo.

20.2.3 Software

Los computadores de la primera generación no disponían de sistema operativo, y la introducción y control de la ejecución de programas se hacía manualmente, uno a uno. En un intervalo de tiempo dado, sólo un programador tenía acceso al computador, lo que implicaba que el

procesador estuviese frecuentemente desocupado, cuando se realizaban transferencias de datos con periféricos de entrada/salida o mientras se cambiaba el turno de un programador a otro (para más detalles véase la Sección 10.3).

Los computadores, en el período 1946 y 1954, se programaban en **lenguaje máquina**, estando, por tanto, cada instrucción constituida por una secuencia de números, con los consiguientes inconvenientes en cuanto a legilibilidad, falta de versatilidad, dependencia del computador, etc. (Sección 6.6). Un hecho a destacar es que el concepto de subrutina ya estaba presente en esta primera etapa, como método de no tener que reprogramar módulos comunes a un mismo programa o a distintos programas (subrutinas *abiertas*).

A comienzos de 1950 se idean los primeros lenguajes simbólicos, con los que las operaciones se pueden representar por nemónicos. Así en 1953 Nathan Rochester diseñó un **lenguaje ensamblador** e implementó el programa correspondiente que traducía instrucciones simbólicas en instrucciones máquina. También se desarrollaron otras ayudas a la programación como generadores de programas (Short Code, Speedcoding, Sort-Merge Generator, Autocode) y los primeros esbozos de compiladores (serie A de Remington Rank).

20.3 Segunda generación de computadores (1955-1963)

Como en la sección anterior vamos a describir brevemente los aspectos tecnológicos, hardware y software más característicos de la Segunda Generación.

20.3.1 Tecnología

Las válvulas termoiónicas ocupaban un gran volumen, necesitaban una energía considerable (recuérdese que cada conmutador dispone de una resistencia de caldeo), y liberaban una cantidad exorbitante de calor, lo que implicaba la necesidad de unas instalaciones muy complejas reguladas en temperatura y humedad. Además el tiempo de vida medio de los triodos era muy limitado, produciéndose con una gran frecuencia averías.

Estos problemas se solucionaron con el **transistor de unión** (Figura 20.9), desarrollado en los Laboratorios de la Bell Telephone en 1948 por J. Bardeen, H. W. Brattain y W. Shockeley. Un transistor está constituido por un pequeño trozo de semiconductor cristalino, al que se le han introducido impurezas de forma controlada. Existen dos tipos de impurezas de **tipo n** (Arsénico, Bismuto, Fósforo, por ejemplo) y de **tipo p** (Aluminio, Galio o Boro, por ejemplo), y al insertarse dentro dela estructura cristalina producen gran cantidad de electrones (de carga negativa) o huecos (de carga positiva), respectivamente. El transistor se realiza con un cristal de Si con dos zonas de tipo n (**emisor** y **colector**), entre las que se ubicaba otra más pequeña de tipo p (**base**), obteniéndose así un **transistor n-p-n** (hay también transistores p-n-p). Habitualmente el circuito de entrada se forma entre el terminal de base y de emisor, y el circuito de salida se forma entre los terminales de colector y emisor. Las cargas del emisor (electrones o huecos, dependiendo de que el emisor sea de tipo n o p, respectivamente) se inyectan hacia el colector a través de la base. El terminal de base controla el paso de corriente en el circuito emisor-colector. Cualquier cambio de tensión entre base y emisor aparece amplificado en el circuito de colector-emisor. Se obtiene así un dispositivo electrónico de estado sólido que se comporta como un triodo (ánodo: colector, cátodo: emisor; rejilla: base), pero mucho más fácil de fabricar (y más barato, por lo tanto), de menor consumo y calor disipado, más pequeño y con una vida media mucho mayor.

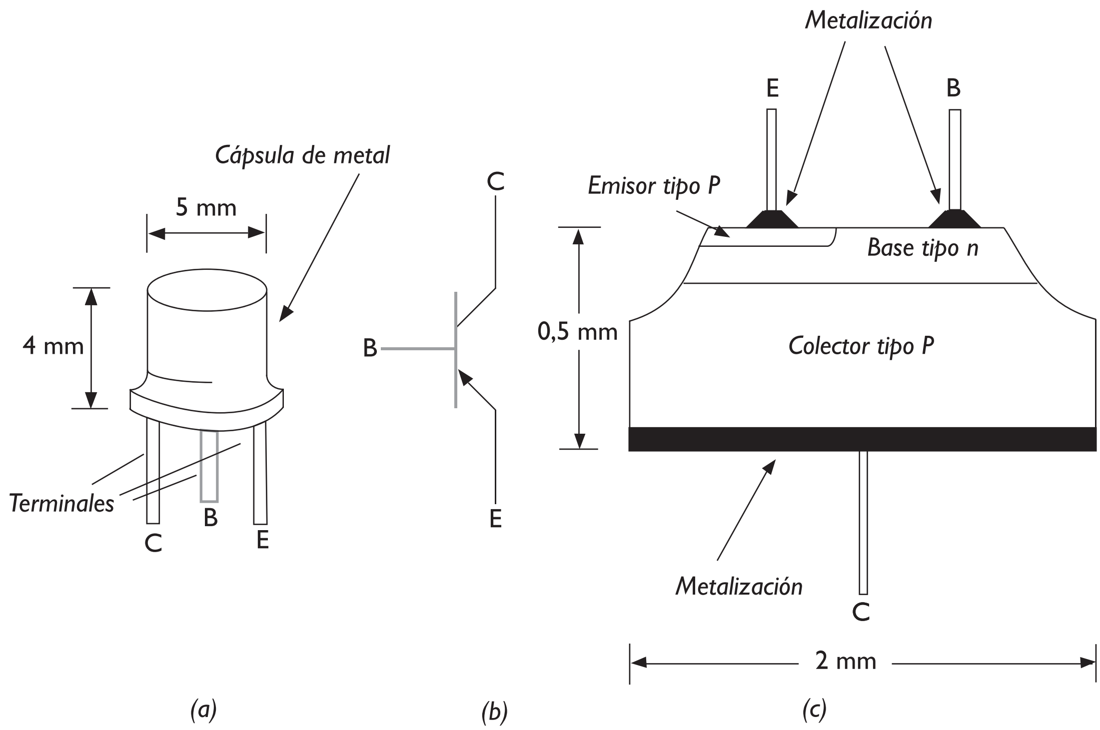


Figura 20.9. *Transistor n-p-n (modelo de 1958): (a) vista exterior; (b) símbolo y (c) corte transversal de su estructura.*

Esta época se caracteriza también por la utilización masiva de **núcleos de ferrita** como elementos básicos de la memoria principal. La introducción de esta tecnología se debe a J. Forrester del M.I.T. (comienzos de la década de los 50). Las memorias de ferrita se forman por paneles o mallas rectangulares de hilos conductores, dispuestos en filas y columnas. En cada vértice se sitúa un núcleo de ferrita, por cuyo interior pasan los hilos correspondientes de la malla. Cada núcleo puede seleccionarse individualmente activando su hilo de fila y su hilo de columna. Al pasar corriente eléctrica por uno de los hilos (hilo de escritura) la ferrita se magnetiza, Norte, por ejemplo, con lo que almacena un bit de información, un 1, por ejemplo; si la corriente pasase en sentido contrario se magnetizaría en sentido contrario, Sur, almacenándose un 0; es decir, cada anillo de ferrita es soporte físico de un bit. Los paneles de ferritas pueden disponerse formando paralelepípedos, de, por ejemplo 8 paneles de altura, las 8 ferritas de una vertical corresponderían a los 8 bits de un byte.

Basándose en la idea de los tambores magnéticos, y buscando una disminución de su volumen, se inventaron los discos magnéticos. Ocupando mucho menos espacio se lograba incluir mucha más información, sobre todo si la unidad se componía de distintos platos girando como un todo. Puede considerarse que el primer disco duro comercializado es el RAMAC (*Ramdom Access Method of Accounting and Control*) introducido por IBM en 1956, y que estaba compuesto por 50 discos de 24 pulgadas, admitía una capacidad de 5 MB, con densidad de grabación de 2 Kbits/pulgada^2, y velocidad de transferencia de datos de 8.800 B/s. Posteriormente, en 1962, también IBM, introdujo el 1301 ADF (*Advanced Disk File*), que disponía de cabezas flotantes,

e incrementaba la capacidad de su predecesor en más de 5 veces, y la densidad de grabación y velocidad de transferencia en un 1.000%.

20.3.2 Computadores

En 1954 se anunció la IBM 650, que fue su primera máquina basada en tambores magnéticos, con los cuales se facilitó notablemente el uso de ensambladores, compiladores y otros programas del sistema.

El primer computador totalmente transistorizado fue el **TRADIC**, construido en 1954 en los laboratorios de la Bell Telephone. Esta máquina, por tanto, puede considerarse que inicia la segunda generación de computadores. Ahora bien, las máquinas más relevantes de esta generación fueron las IBM 7090 y 7094, que dominaron el campo de las aplicaciones científicas durante la década de los años 60. Esencialmente eran versiones transistorizadas del IBM 704 y su sucesor, el IBM 709. Las versiones transistorizadas (al igual que el 709) disponían de **canales de E/S** (posteriormente denominados **procesadores de E/S**), que eran procesadores de uso específico para controlar las operaciones de entrada/salida.

También en esta época se generalizó el uso de registros índices y de hardware para coma flotante.

Otros sistemas significativos de esta etapa fueron el UNIVAC 1004, IBM 1620 y CDC 1604.

20.3.3 Software

Aunque los lenguajes ensambladores son mucho más fáciles de usar que los lenguajes máquina, el tipo de instrucciones de que disponen sigue muy alejado del tipo de problemas reales que se pretenden resolver. Además, distintos computadores tienen distintos lenguajes ensambladores siendo prácticamente nula la transportabilidad de los programas: un programa en lenguaje ensamblador realizado para un procesador no funciona en otro distinto. A mediados de la década de 1950 se empezaron a introducir **lenguajes de alto nivel**, que permitían escribir programas en una forma más próxima a la naturaleza del problema a resolver que al conjunto de instrucciones que interpreta directamente el procesador (instrucciones máquina). Estos lenguajes, además, podían utilizarse en distintas máquinas.

De 1954 a 1957, bajo la dirección de John Backus de IBM, se desarrolló el primer lenguaje de programación de amplia difusión: el FORTRAN (FORmula TRANslator, o *traductor de fórmulas*), proyectado para la programación de aplicaciones de cálculo científico y técnico. El FORTRAN permite la redacción de expresiones algebraicas en una notación muy próxima a la notación algebraica convencional. No hay que olvidar que los primeros computadores se desarrollaron pensando en la resolución de problemas de tipo matemático.

Después del FORTRAN se desarrollaron varios lenguajes de importancia, como en 1958 el ALGOL58 (ALGOrithmic Language) y su sucesor el ALGOL60 (1960) que fue un lenguaje con una gran aportación conceptual por lo que sirvió de base para el desarrollo de muchos lenguajes de uso en la actualidad, LISP, ideado por John McCarty en 1962 y muy adecuado para operar con expresiones simbólicas y para aplicaciones de inteligencia artificial, y SNOBOL, producido para aplicaciones de tratamiento de caracteres, en el Bell Telephone Lab. por David J. Faber, Ralp E. Griswold e Ivan Polonsky (1962). En la Figura 20.9 se esquematiza la evolución histórica de los lenguajes de programación pioneros y que más han influido en los lenguajes actualmente utilizados.

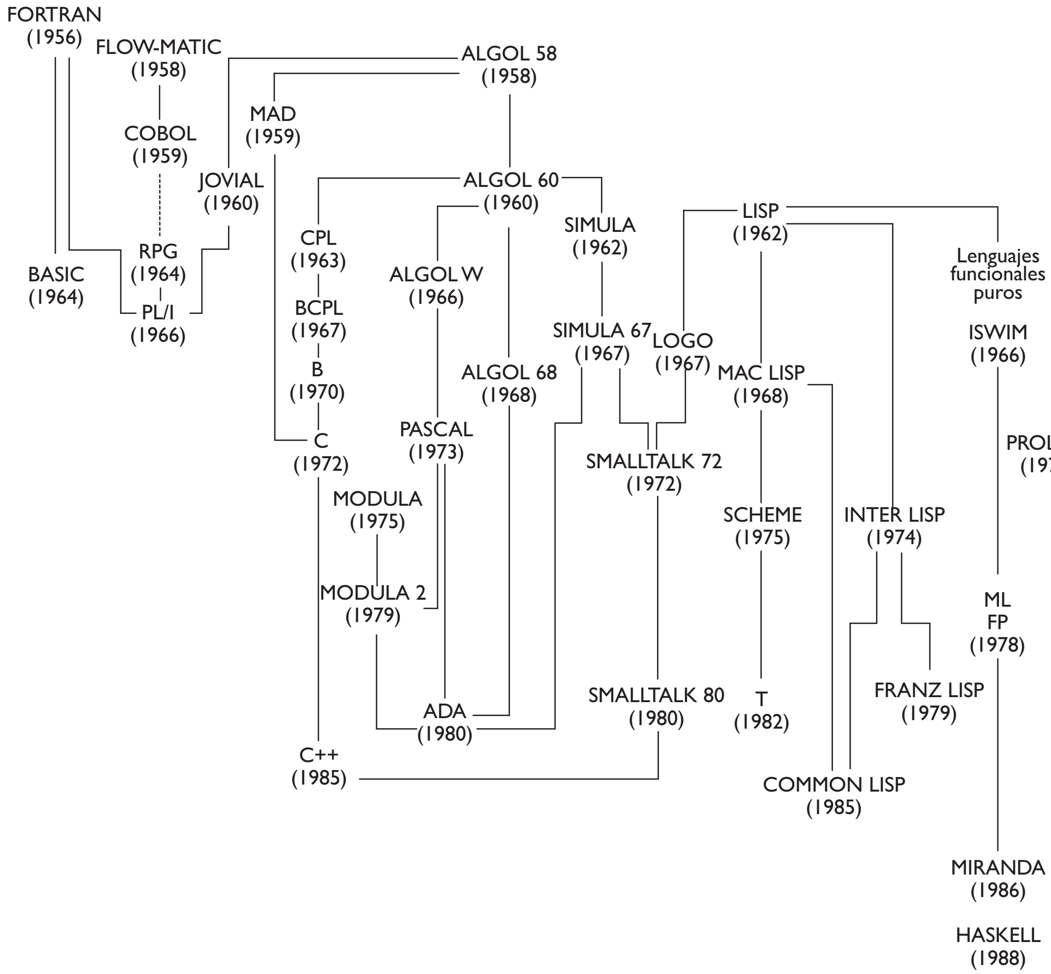


Figura 20.10. *Esquema que muestra la evolución de los lenguajes de alto nivel pioneros.*

También se desarrollaron lenguajes de programación de alto nivel para aplicaciones de gestión administrativa. Se caracterizan por utilizar sentencias que, con nemónicos ingleses, disponen de las funciones matemáticas típicas de gestión (funciones financieras, etc.) y actúan fundamentalmente con archivos de información alfanumérica. El lenguaje de mayor incidencia fue el COBOL (COmmon Bussines Oriented Language), que fue definido en 1959 por la Conferencia en Data Systems Languages (CODASYL).

Hemos visto que en la primera generación de computadores, cada trabajo o programa se procesaba independientemente. Cuando acababa un programa, se detenía la máquina, y se debía preparar manualmente para la traducción, o carga y ejecución del siguiente. Con las mejoras hardware de los sistemas de entrada/salida, fue posible, mientras el computador procesaba un trabajo, preparar nuevos trabajos y grabarlos (con frecuencia con ayuda de un pequeño computador periférico) en una cinta magnética uno detrás de otro, formando una **cola de trabajos** (procesamiento *batch*) en la cinta. Una vez que una cinta contenía una cola de trabajos, se

ubicaba en una unidad de entrada del computador central, y ésta, por medio de un **programa supervisor** o **monitor**, que residía permanentemente en memoria, iba cargando programa tras programa tan pronto como pudiese irlos ejecutando. Esto supuso el nacimiento de los sistemas operativos. En la Sección 13.4 se incluyó una descripción más detallada del nacimiento y evolución de los sistemas operativos (Figuras 13.2 y 13.3).

En esta generación los fabricantes de computadores suministraban gran cantidad de software tal como compiladores, bibliotecas de subrutinas, y programas de supervisión más o menos sofisticados (sistemas operativos).

20.4 Tercera generación de computadores (1964-1970)

A continuación describimos los hitos más importantes de la tercera generación de computadores.

20.4.1 Tecnología

En 1958, Kilby de Texas Inst., concibió la idea de realizar un **circuito integrado monolítico**; esto es, la construcción de un circuito entero en un substrato de Ge o Si. En distintos substratos semiconductores construyó una resistencia, un condensador, transistores, etc. Las conexiones entre los distintos elementos las realizó por termo-compresión con hilos de oro; pero ya en su patente indicaba la posibilidad de que los componentes podrían ser integrados en un único substrato de semiconductor y las interconexiones sugería se efectuasen depositando sobre la superficie pistas o tiras de material conductor. En 1960, en un congreso, anunció su idea sobre los *circuitos sólidos* más adelante denominados **circuitos integrados**. Los primeros desarrollos comerciales de éxito se deben al desarrollo de la **tecnología planar**, propuesta por Noyce y Moore, consistente en la fabricación de dispositivos utilizando procesos fotolitográficos y de difusión térmica. Esta propuesta posibilitó la comercialización de circuitos integrados a partir de 1961 por Fairchild y Texas Inst. La tecnología integrada hace posible la construcción, en un sólo bloque de semiconductor, de un circuito que realice una función electrónica compleja (puertas lógicas, biestables, etc.). Los distintos elementos quedan conectados directa e inseparablemente, obteniéndose sobre el substrato de cristal una especie de microfotografía del circuito. Algunos de los hechos más relevantes de la nueva tecnología son los siguientes:

- El bajo costo de los transistores que incluye el circuito integrado hizo posible diseños de circuitos cada vez más complejos y perfectos.
- Al estar todos los componentes de un circuito muy próximos, en el mismo cristal, los retardos en la transmisión de señales entre distintos puntos son mínimos.
- Se reduce el consumo de energía y la fiabilidad de los circuitos.
- Se consigue una gran miniaturización.
- Se reduce considerablemente el costo, ya que:
  - Se puede automatizar considerablemente la construcción de los circuitos, pudiéndose fabricar en grandes series.
  - Los nuevos dispositivos no sólo contienen conmutadores, sino circuitos completos; es decir, se reduce considerablemente el trabajo manual de interconexión de componentes, se construye simultánea y automáticamente todo el circuito.

En la Tabla 20.3 se muestra una clasificación de los circuitos integrados en cuanto a su complejidad o grado de integración.

Tabla 20.3. *Clasificación de los circuitos integrados según su grado de integración.*

Año	Grado de integración	Nº de transistores	Funciones integradas
1961	SSI: pequeña escala	4 a 100	Puertas lógicas, biestables, etc.
1966	MSI: media escala	100 a 1.000	Circuitos aritméticos, registros, contadores, etc.
1971	LSI: gran escala	1.000 a 10.000	Memorias, microprocesadores de 8 bits, etc.
1980	VLSI: muy gran escala	20.000 en adelante	Microprocesadores, DRAM, Microcontroladores, etc.
1999		1.000.000 en adelante	
2000		10.000.000 en adelante	

Recuérdese que el ENIAC contenía aproximadamente 20.000 conmutadores (18.000 válvulas de vacío y unos 1.500 relés), con lo que todos sus circuitos hubiesen podido integrarse fácilmente en un circuito VLSI; de hecho la tecnología actual de fabricación de circuitos integrados permite integrar toda la electrónica que contenía el ENIAC en un volumen que puede pasar con creces por el ojo de una aguja. Por otra parte, cualquier calculadora electrónica de bolsillo actual realiza funciones más complejas, más rápidamente y con más precisión que el ENIAC, y ¡sin necesitar una superficie de unos 140 m^2!

Tecnológicamente la tercera generación de computadores se caracteriza por la utilización de circuitos integrados SSI y MSI, reemplazando a los circuitos de transistores discretos, y consiguiéndose así una reducción significativa en costo y tamaño físico. Poco a poco también se fueron imponiendo las memorias realizadas con circuitos integrados, desplazando a las memorias de núcleos de ferrita.

En cuanto a dispositivos de memoria externa cabe destacar la comercialización de los discos flexibles (IBM, 1967); apareciendo años después las normalizaciones más utilizadas: 8” (IBM, 1971), 5¼” (Shugart Associates, 1976), y 3½” (Sony, 1980).

20.4.2 Computadores

En esta etapa se desarrollan grandes computadores, que daban servicio a una gran diversidad de terminales trabajando en forma local o remota. Para ello se desarrollaron procedimientos para compartir los recursos del computador (procesador, memoria, periféricos, etc.). Cabe destacar el desarrollo de la **memoria virtual**, técnica que permitía al usuario disponer de una memoria mayor que la capacidad física real de la memoria principal (Sección 9.6.5).

Puede considerarse que la consolidación del diseño de unidades de control microprogramadas, tal como las concibió Wilkes años antes (1951, véase Sección 20.2) se debe a su introducción en algunos modelos de la **serie IBM 360** (aparecidos a mediados de los 60).

También se perfeccionó y se generalizó el uso de una gran variedad de técnicas para procesamiento concurrente o paralelo, tales como segmentación de cauce (*pipelining*), multiprogramación y multiprocesamiento.

Puede considerarse que el computador más significativo de la época fue el **IBM 360**, citado anteriormente y comercializada a partir de 1965. Uno de los aspectos más destacables de la 360, es que se concibió, más que como un computador concreto, como una *línea* o *familia* de computadores utilizable tanto para aplicaciones comerciales como científico-técnicas; en

efecto, todos los modelos de la serie tenían el mismo repertorio de lenguaje máquina (es decir, teóricamente los programas eran compatibles) y el mismo sistema operativo, difiriendo tan solo en precio y prestaciones (memoria máxima, velocidad, número de E/S, periféricos permitidos, etc.), debido a diferentes implementaciones físicas.El modelo inferior de la serie era comparable al computador más pequeño de IBM de la generación anterior (1401, de aplicaciones comerciales) y el modelo superior (IBM 360-91) era mucho más capaz que el 7090 (gran computador de aplicaciones científicas). Algunos modelos de la serie 360 difundieron por primera vez nuevas técnicas tales como microprogramación, segmentación de cauce, memoria caché (IBM 360/85), canales de E/S, controladores DMA, etc., y fue el primer computador comercial que utilizó circuitos integrados (SSI).

Otros equipos destacados de esta generación fueron el UNIVAC 1108 y el IBM 370. Mención aparte merecen los supercomputadores escalares diseñados por el equipo de Seymour Cray en Control Data Corporation, la **CDC 6600** (1963-1964), la**CDC7600** (1969), y la serie **CDC CYBER**; que conseguían unas prestaciones de pico del orden de 15 a 20 Mflop/s (millones de instrucciones de coma flotante por segundo). EL CDC 6600 fue revolucionario en muchos aspectos y es considerado como el primer supercomputador y uno de los primeros con paralelismo a nivel de instrucción. Su diseño se basó en la utilización de paralelismo funcional, de forma que disponía de 10 unidades aritméticas no segmentadas separadas funcionalmente, 32 bancos independientes de memoria, y 10 procesadores de entrada/salida.

En el extremo opuesto, en cuanto a precio y prestaciones, en 1963 la empresa Digital Equipment Corporation (DEC) comenzó a comercializar una serie de computadores denominados PDP (1963: **PDP-5**, 1965: **PDP-8**, de la que se vendieron 50.000). Este tipo de máquinas se denominaron **minicomputadores**, y eran únicos en muchos aspectos. Se trataba de comercializar pequeños equipos informáticos (longitud de palabra reducida: 16 bits, número y tipo de periféricos muy limitado, etc.), que aprovechasen al máximo las posibilidades de la tecnología de circuitos integrados. Su precio era al menos un orden de magnitud inferior al de los computadores convencionales de la época y su campo de aplicaciones era muy amplio, pues iba desde satisfacer las necesidades de un pequeño Departamento de una Universidad o empresa, hasta ser utilizado en control industrial, como computador de uso específico dedicado en exclusiva a una aplicación. Las prestaciones de estos equipos llegaron a ser comparables a las de los mayores computadores de la segunda generación. El **PDP-11** (1970) disponía de un bus normalizado al que podían conectarse tarjetas de distintos fabricantes, siendo por ello precursor del concepto de *sistema abierto,* al que nos referiremos en la Sección 20.5.

20.4.3 Software

Una de las características más destacables de la Tercera Generación de computadores fue el desarrollo de grandes computadores atendiendo a un gran número de terminales. Se difundieron sistemas operativos que utilizaban técnicas tales como multiprogramación, multiprocesamiento, secuencias múltiples, dispositivos de entrada/salida virtuales (*spool*), etc. (véase Capítulo 10). Cada computador (o serie compatible de ellos) se utilizaba con un sistema operativo específico diseñado por el propio constructor del hardware.

En la segunda mitad de la década de los 60 se idearon nuevos lenguajes de programación de alto nivel tales como el BASIC (Thomas E. Kurtz y John G. Kemeny, 1964), PL/I (IBM, 1966), APL (Desarrollado por Kenneth E. Iverson entre 1957 a 1960), el Pascal (Niklaus Wirth, 1970-73). También se proyectaron nuevas versiones de lenguajes precedentes (Figura20.9).

20.5 Cuarta generación de computadores (1971-...)

Una de las peculiaridades de la actual generación de computadores (cuarta) es el concepto de *sistema abierto*. En las generaciones anteriores cada computador estaba muy ligado a su fabricante (**sistema propietario**), de forma que tanto el software como los distintos elementos hardware, incluidos los periféricos, eran suministrados, en la configuración inicial y en ampliaciones sucesivas, *exclusivamente* por el fabricante original. Con los **sistemas abiertos** la filosofía es distinta: el usuario puede configurar su sistema prácticamente a medida, adquiriendo software, impresoras, unidades de disco, etc. de "terceros" fabricantes. Obviamente el desarrollo de los sistemas abiertos implicó la definición de numerosas normalizaciones o estándares que debían seguir los distintos fabricantes para introducir en el mercado adecuadamente sus productos. También la presente generación se caracteriza por los logros que se indican a continuación.

20.5.1 Tecnología

Se utilizan circuitos integrados LSI y VLSI, lo que permitió (año 1995) la comercialización de circuitos integrados de memoria conteniendo 1 Gbits, con tiempos de acceso de 35 ns, y el desarrollo de los microprocesadores y microcontroladores (Secciones 1.2.1 y 5.3.4), que permitieron la difusión en gran escala de la informática (aplicaciones de control, electrodomésticos, vehículos, etc.). En las Tablas 5.5 a 5.8 se enumeran algunos de los microprocesadores más conocidos. Grosso modo puede decirse que la década de 1970 es la de los microprocesadores de longitud de palabra de 8 bits, la de los 80 la de 16/32 bits, y la de los 90 la de 32/64 bits. Al inició de la década de los 2000 se superaron las frecuencias de reloj de 1 GHz para el procesador.

Muchas funciones hardware de gran complejidad se implementan con circuitos integrados, con las consiguientes ventajas en cuanto a precio, miniaturización, ahorro de consumo y fiabilidad: controladores DMA, gestores de memoria, gestores de memoria virtual, controladores de E/S, coprocesadores aritméticos, etc. Poco a poco estos elementos fueron incluyéndose dentro del propio chip del microprocesador.

Cabe destacar la miniaturización y aumento de velocidad de las memorias de circuito integrado. En 1970 Intel comercializa las primeras RAM dinámicas (DRAM), y desde entonces se ha logrado un incremento de su capacidad constante: se multiplica por 4 su capacidad cada 3 años. Así en 1980 se comercializaron chips de 64 Kb con tiempos de acceso del orden de 250 ns, ya en 1992 había chips de 16 Mb con tiempos de acceso de 90 ns, y en 1999 Samsung desarrolló una memoria DRAM de 1 Gbit capaz de operar a 350 MHz, utilizando una litografía de 0,13 μm. Las previsiones apuntan a chips de memoria de hasta 64 Gbits hacia las proximidades de 2010 utilizando litografías en torno a 0,05 μm.

En cuanto a dispositivos de memoria externa, en 1973 IBM introdujo el modelo 3340, antecesor de los discos duros actuales. Una unidad 3340 constaba de dos sub-unidades independientes, una fija y la otra intercambiable, con una capacidad total de 30 + 30 MB. Por conocerse como una unidad "30-30", al igual que ocurría con el afamado rifle de repetición Winchester 30-30, esta tecnología se denomina **tecnología Winchester** (Sección 8.3.3.5).

En 1979 la empresa fabricante de discos duros Shugart Associates inició el desarrollo de una nueva interfaz, posteriormente denominada SCSI (*Small Computer Systems Interface*), que permitiera utilizar direccionamiento lógico por bloques de datos, en lugar de utilizar direccionamiento físico (cilindro, superficie, sector). En 1986 se aprobó el primer estándar conocido como **SCSI-1**, y en 1989 la versión mejorada conocida como **SCSI-2** (Sección 8.4).

Seagate, en 1980, comercializa el primer disco duro de 5¼”, denominado ST-506. El disco tenía una capacidad de 5 MB y disponía 4 cabezas. Una versión posterior, ST-412, de 10 MB de capacidad, fue el elegido por IBM para el IBM PC/XT, pasando a ser considerado un estándar de hecho dada su amplia difusión.

Por otra parte, en 1985, a petición de Compaq, la empresa Imprimis desarrolló una unidad de disco duro que incluía un controlador ST-506. Este hecho marcó el nacimiento de los discos **IDE**, uno de los estándares más utilizados en la actualidad (aprobado en 1994 con el nombre de **ATA**).

La primera unidad de discos magneto-ópticos (Sección 8.3.6) fue desarrollada por Sony en 1988; el mismo año en que surge la tecnología de los discos flópticos (Sección 8.3.3) y se normalizan los niveles RAID (Sección 8.3.3.7).

20.5.2 Computadores

Los aspectos de mayor relevancia son la comercialización de los computadores personales (PC), que supuso la difusión plena, incluso a nivel familiar, de la Informática. El primer PC, el Altair, se produjo en 1975 por MITS de Albuquerque. No obstante, el predecesor más significativo de los actuales sistemas PC se encuentra en el Apple II (1977) que se diseñó con los criterios de miniaturización, bajo coste, alta fiabilidad, y mejoresprestaciones posibles para un entorno monousuario. El IBM-PC se anunció en Agosto de 1981 (Sección 5.7). También es característico de esta generación el concepto de estación de trabajo, que en cierta medida puede considerarse el minicomputador de la cuarta generación en cuanto a sus objetivos: altas prestaciones a muy bajo precio atendiendo demandas departamentales para aplicaciones científico-técnicas. Puede considerarse que los orígenes de las estaciones de trabajo se remontan a 1973, cuando en Xerox (Palo Alto) desarrollaron un microcomputador (el Alto) con grandes posibilidades gráficas (ventanas, uso del ratón, etc.) junto con otros elementos característicos del uso de este tipo de equipos: red local Ethernet, impresora láser, servidor de archivos, etc.

Un hecho notable ha sido el desarrollo de la telemática, que permite la creación de grandes redes de computadores, tanto de área local como de áreas públicas, distribuyendo eficientemente los recursos tanto hardware como software.

En cuanto a avances arquitectónicos es de destacar el surgimiento (Patterson, Universidad de Berkeley 1980; Hennessy Universidad de Stanford 1981; John Cocke de IBM 1982) de procesadores de **conjunto reducido de instrucciones** (**RISC**) (Sección 5.3.5), en los que, además de tener un repertorio de instrucciones máquina muy limitado, se tiende a que todas éstas sean de longitud fija, con pocos modos de direccionamiento y los transvases entre procesador y memoria sólo se realizan con instrucciones específicas de carga-almacenamiento y operaciones sólo con datos en registros (de los que se dispone en cantidad) para facilitar la optimización de la segmentación de cauce. **MIPS** fue el primer computador RISC comercial (1985).

También es de destacar la comercialización de supercomputadores de distintos tipos, como vectoriales, matriciales y multiprocesadores. Enla Tabla 20.4 se incluyen algunos ejemplos ilustrativos, siguiendo la clasificación efectuada en la Sección 5.8. El **Cray-1** (1976) fue el primer supercomputador vectorial, y la máquina más cara y rápida del mundo en su momento. Fue construida en la empresa Cray Research, fundada por Cray. En 1985, con la comercialización del Cray 2 y la Connection Machine (de Thinking Machines), se obtienen por primera vez supercomputadores que alcanzan tasas de ejecución de mil millones de operaciones por

segundo. En 1990 se comercializó el IBM **RSC6000** (o Power1, predecesor del **PowerPC**), que es considerada como la primera máquina superescalar; con estas máquinas se pueden ejecutar varias instrucciones a la vez, no necesariamente en el mismo orden en que se encuentran en el programa.

Tabla 20.4. *Supercomputadores representativos.*

	Tipo	Velocidad de pico	Ejemplos y suministradores
SISD	Escalares (3ª Generación)	15-20 Mflop/s	CDC 6600-7600 IBM 360-91
SIMD	Matriciales	20 Gflop/s	Illiac IV, MPP; IBM GF/11; CM2, BSP, ICL-DAP, Cray-3/SSS, MP-1216
	Vectoriales	10-30 Gflop/s	Cray-1, Cyber-205, Cray X-MP; Convex C-1; Fujitsu VP 100; CDC-ETA; Cray T90; Fujitsu VPP300
MIMD	Multiprocesado-res con memoria compartida	Gflops	C.mmp; Cedar; IBM RP3; Fujitsu VPP500, Dash, TC-2000; KSR-1 *Conectados por bus único:* IBM RS/6000 R40;Sun Enterprise 6000; Digital AlphaServer 8400 *Conectados con red:* Cray R.T3E; HP/Convex Exemplar X-Class; SGI Origin 2000; Sun Enterprise 10000
	Multiprocesado-res con memoria distribuida (Mul-ticomputadores)	0,1-100 Tflop/s	Cosmic Cube, Intel iPSC; Intel Parangon; nCUBE; Mosaic-C; J-Machine HP 9000 EPS21; Sun Enterprise Cluster 6000 HA; IBM RS/6000 SP2; Supercomputadores ASCI

En marzo de 2006 el supercomputador más rápido del mundo, como indicamos en la Sección 1.5 es el BlueGene/L – eServer construido en el 2005 por IBM y ubicado en el Lawrence Livermore National Laboratory de la Agencia de Seguridad Nuclear Nacional del Departamento de Ebergía de los Estados Unido. Contiene 131.072 procesadores PowerPC, y alcanza una tasa de ejecución de pico de 367 TeraFlops. El sistema se encuentra instalado en una gran sala dentro de 64 armarios metálicos.

20.5.3 Software

En el comienzo de esta generación se anuncia el primer computador con memoria virtual (IBM-370, año 1972).

Una de las características fundamentales de la presente generación es la difusión de sistemas operativos estándar, no dependientes directamente de los constructores del hardware y, por tanto, utilizables por distintos modelos de computadores. Los casos más sobresalientes son el UNIX (Ken Thompson y Dennis Ritchie, Bell Labs. 1974), proyectado inicialmente para minicomputadores (DEC PDP-7) y en la actualidad muy utilizado para estaciones de trabajo y computadores superiores, el MS/DOS, para computadores personales, y cuya versión 1.0 se comercializó en Agosto de 1981 (Tim Paterson), el OS/2 (IBM y Microsoft, Abril 1988), y el Windows NT de Microsoft (Agosto 1993). También merece mención el caso del Linux, que surgió en 1991 como proyecto de Linus Torvalls, alumno de informática de la Universidad de Helsinki (Finlandia). En la actualidad el Linux puede considerarse como un clónico del UNIX

utilizable en muy distintas plataformas, incluyendo PC. En la Sección 9.4.2 se analizó con cierto detalle la evolución de los sistemas operativos mencionados anteriormente (Figuras 9.2 y 9.3).

También es de destacar el desarrollo (a mediados de los 80) de sistemas operativos de red y sistemas operativos distribuidos. Con un **sistema operativo de red**, un usuario de un computador puede realizar operaciones tales como introducirse a través de la red en un computador remoto, ejecutar programas allí, o copiar archivos de un computador a otro. Un **sistema operativo distribuido** permite al usuario trabajar aparentemente con un único computador, siendo el sistema operativo el que automáticamente distribuye (de forma "transparente" para el usuario) su programa o programas y archivos en diferentes computadores. Por ejemplo, se puede consultar un dato a una base de datos distribuida entre varios computadores y el sistema operativo puede encaminar adecuadamente la consulta y su respuesta.

A partir de 1971 se consolidan conceptos nuevos de programación, tales como el de programación lógica, y el de programación dirigida a objetos que supone una herramienta que facilita notablemente el desarrollo de las facultades creativas del programador. Cabe destacar el desarrollo del lenguaje de programación lógica PROLOG (Alain Colmerauer, Phillipe Rousse y Rober Kowalski, de las Universidades de Marsella y Edimburgo, 1972) que es especialmente útil en aplicaciones de lenguaje natural, demostración automática de teoremas, construcción de sistemas expertos y gestión y consulta de bases de datos inteligentes. El lenguaje más característico para la programación dirigida a objetos es el SmallTalk (Alen C. Kay, Xerox, 1972).

Otro lenguaje de gran interés es el C (1972, M. Ritchie y K. Thonpson, Bell Telephon Lab.), que se ideó con objeto de facilitar la redacción de las primeras versiones de UNIX. Su principal ventaja radica en que es de una gran portabilidad y además de ser un auténtico lenguaje de alto nivel, permite codificar operaciones a muy bajo nivel (manipulación directa de registros del procesador, etc.).

También merece mención el lenguaje Ada, que fue definido por comités de expertos designados por el Departamento de Defensa (DoD) de los Estados Unidos, al objeto de integrar las posibilidades de todos los lenguajes importantes de la época, y unificar así las herramientas de programación de todo el Departamento, al objeto de reducir costes, ya que se estimó que los gastos en software del DoD para el año 1990 sería de unos 30.000 millones de dólares. Fue implementado por primera vez en 1980 por un equipo dirigido por Jean Ichbiah de CII-Honeywell-Bull. Bajo una perspectiva histórica se puede concluir que los esfuerzos dedicados al desarrollo del Ada fueron inútiles ya que este lenguaje no cubrió ni mínimamente las expectativas buscadas.

A principios de la década de 1980 B. Stroustrup de los Laboratorios Bell desarrolló el lenguaje C++. Puede considerarse como un evolucionado del C, ya que es igual de potente y eficiente, pero además lleva a la práctica los conceptos que subyacen tras la programación orientada a objetos (Sección 10.4). Sun Microsystems en 1991 desarrolló otro lenguaje, Java, inspirado en C++ y de gran éxito en la actualidad para programación orientada objetos. La gran ventaja de este lenguaje, además de incluir la mayoría de las modernas técnicas de programación y de seguridad en redes, es que es utilizable en cualquier computador que disponga de un programa emulador de un computador virtual denominado JVM (*Java Virtual Machine*). De esta forma se consigue una compatibilidad total de programas en código binario, entre máquinas distintas que dispongan del intérprete JVM. La mayor parte de los navegadores de internet disponen de un emulador de JVM para poder ejecutar pequeños programas (*applets*) binarios Java relacionados a paginas web (de animación y sonido, por ejemplo).

20.6 Las generaciones en el desarrollo de los lenguajes de programación

En la Figura 20.10 incluimos un esquema que muestra la evolución de los primeros lenguajes de programación. También en la Figura 14.9 se dio una lista de los lenguajes más usados, su año de nacimiento, y su ámbito de aplicación.

Según se indicó al comienzo de la Sección 20.2 el concepto de generación de computadores (que se refiere principalmente al hardware) no coincide con el dado por los especialistas para los lenguajes de programación.

Usualmente se conocen como lenguajes de primera generación a los lenguajes máquina (hasta 1952). En la segunda generación (1949 a 1956) se suelen incluir las primeras ayudas a la programación tales como rutinas interpretativas, lenguajes ensambladores, y generadores de programas (a las que nos hemos referido en la Sección 20.2).

Se conocen como lenguajes de la tercera generación (a partir de 1954) los **lenguajes imperativos** (Capítulo 10) o lenguajes de alto nivel clásicos: FORTRAN, COBOL, LISP, SNOBOL, APL, BASIC, Pascal, C y Ada.

En la cuarta generación (1961 a la actualidad) se incluyen los **lenguajes funcionales**, en su doble vertiente de **lenguajes de programación lógica** (Prolog) y **lenguajes de programación dirigida a objetos** (Smalltalk, C++, Java, etc.). No obstante lo anterior, vulgarmente se conoce como **lenguajes de cuarta generación,** o **4G**, a lenguajes o entornos de programación para usos específicos (dBASE, ORACLE, FOCUS, Visual Basic, etc.), que sirven de puente para aproximar cada vez más el computador a la aplicación y al lenguaje natural, y con los que se pueden programar aplicaciones rápidamente. Una declaración en un lenguaje de cuarta generación es equivalente a un gran número de sentencias de un lenguaje imperativo, perdiéndose flexibilidad pero ganándose en eficacia al poder desarrollar las aplicaciones con gran rapidez. Dentro de los lenguajes de 4G pueden incluirse los lenguajes visuales, como Visual Basic o Delphi. Estos lenguajes permiten desarrollar la interfaz gráfica de una aplicación con el usuario (GUI) manualmente con ayuda del ratón,seleccionando de una *caja de herramientas* componentes predeterminados (cuadros, botones, iconos, cajas de texto, etc.), regulando su tamaño y posición, etc. Una vez diseñada y visualizada la interfaz gráfica y las acciones asociadas a cada elemento, puede generarse automáticamente el código correspondiente, que establece la forma exacta, tamaño, ubicación, etc. de los objetos y datos a visualizar, e incluirlos en la aplicación.

La tendencia actual más relevante de los lenguajes de programación es tratar de desarrollar entornos de programación y programas lo más cómodos posibles de utilizar (amigables) y como consecuencia lo más próximo posible al lenguaje natural.

20.7 Resumen de los hitos más destacados en el desarrollo de la informática

En la Tabla 20.5 incluimos una tabla resumen de algunos de los desarrollos más notables de la historia y evolución de la informática. Más detalles, ilustrados con fotografías, pueden encontrarse en [Car96].

Tabla 20.5. *Resumen de algunos de los hitos más destacados en el desarrollo de la informática.*

Fecha	Investigador	Desarrollo	Descripción
1936	K. Zuse (Alemania)		Propone realizar programas con combinaciones de bits que pueden ser almacenadas. Patenta una aplicación para la ejecución automática de cálculos
1938	Honrad Zuse (Alemania)	Z1	Concluye un computador binario electromecánico
1939	J. Atanasoff y Cl. Berry	ABC	Construye un prototipo de un computador electrónico digital que usa aritmética binaria
1943	A.Turing, T. Flowers, M. H. Newman	Colossus	1er dispositivo de cálculo completamente electrónico
1944	H. Aiken y G. Hopper	Harvard Mark I	Calculador digital capaz de realizar las 4 operaciones básicas en una secuencia predeterminada
1945	J. von Neumann	EDVAC	Introduce el concepto de programa almacenado y esboza el diseño del EDVAC
1946	J. P. Eckert y J. W. Mauchly	ENIAC 1	1er computador electrónico de uso general, con más de 20.000 tubos de vacío
1947			Se introducen los tambores magnéticos como memoria masiva
1947	J. Bardeen, W. Brattain, W. Shockley	Transistor	Bell Labs anuncia el desarrollo del primer transistor. (23/12/1947)
1948	Manchester	Mark I	1er computador digital funcionando de programa almacenado
1949	J. Forrester (MIT)	Whirlwind	1er computador para tiempo real
1951	J. P. Eckert y J. W. Mauchly	Univac I	1er computador commercial (Oficina de censos de EEUU)
1951	J. Forrester	Matrices de núcleos de ferrita	Nuevo sistema de memoria
1951	M. Wilkes		Introduce el concepto de microprogramación
1952	G. M. Hopper	A-0	1er compilador
1954	Univac	Univac 1103A	1er computador con memorias de ferrita
1956	Univac		1er computador de transistores
1957	John Backus (IBM)	FORTRAN	1er lenguaje de alto nivel ampliamente difundido (realizado para Westinghouse)
1958	Jack Kilby (Texas Instr.), y R. Noyce (Fairchild Semic.)	Chips	Prototipos de Circuitos Integrados
1959	Comite Codasyl	COBOL	Se crea el Comitte on Data Systems Language para definer el lenguaje COBOL
1959	J. McCarthy	Lisp	Lenguaje de alto nivel para aplicaciones de inteligencia artificial
1960		Algol 60	Se establece por científicos americanos y europeos el Algol 60, como un lenguaje estándar

Tabla 20.5. *Resumen de algunos de los hitos más destacados en el desarrollo de la informática. (Cont.)*

Fecha	Investigador	Desarrollo	Descripción
1960	Digital	PDP-1	1er computador con un monitor de salida y teclado de entrada
1964	IBM	Sistema/360	Anuncio del primer computador de la tercera generación
1964	J. Kemeny, T. Kurtz	BASIC	Desarrollo del lenguaje de programación Basic
1964	Control Data Corp.	CDC	1er supercomputador, 9 MegaFlops
1964	Douglas Engelbart	Ratón y ventanas	Invención del ratón y de la presentación de la información por medio de ventanas
1965	Digital Equipment Corp. (DEC)	PDP-8	1er minicomputador. Realizado con tarjetas-módulos de transistores
1965	M. Wilkes		Concibe el concepto de caché
1967	O. J. Dahl y K. Nygaard	Simula	1er lenguaje de programación de uso general orientado a objetos
1968	Burroughs	B2500 y B3500	1er computador con circuitos integrados
1968	Rand Corp.	ARPA	Se presenta el concepto de red de comunicaciones descentralizado, fundamento de Internet
1970	D. Ritchie y K. Thomson (Bell Lab.)	Unix	Desarrollo del sistema operative Unix
1970	RCA	MOS	Los circuitos MOS se presentan como tecnología para realizar chips más baratos y pequeños
1970	Intel	Intel 1103	1er chip de memoria RAM (dinámica)
1971	Ted Hoff, S. Mazor y F. Fagin (Intel)	Intel-4004	1er microprocesador
1971	A. Shugart (IBM)	Disquete	
1971	Ray Tomlinson (Bolt Beranek and Neuman)		Envío del 1er mensaje e-mail
1972	D. Ritchie (Bell Labs.)	Lenguaje C	
1972	A. Colmerauer (Unv. Marsella)	Lenguaje Prolog	Lenguaje de programación lógica
1973	Xerox PARC	PC Alto	1er computador personal experimental
1973	R. Metcalfe	Ethernet	Concepción de la red local Etrhernet
1975	Altair	PC	1er PC. Contenía un 8800 y se vendía sin montar
1976	Cray Research	Cray-1	1er supercomputador de arquitectura vectorial
1976	G. Kildall	CP/M	Sistema operativo para PC de 8 bits
1976	IBM	Impresora de inyección de tinta	

Tabla 20.5. *Resumen de algunos de los hitos más destacados en el desarrollo de la informática. (Cont.)*

Fecha	Investigador	Desarrollo	Descripción
1978	DEC	VAX 11/780	Minicomputador de 32 bits de un gran éxito para aplicaciones científico-técnicas
1978	Dan Bricklin & Bob Frankston	VisiCalc	1er software para hoja electónica
1978	Intel	Intel 8086	1er microprocesador de 16 bits
1979	Sony y Philips	DVD	Comercialización de los videodiscos digitales (DVD)
1979	S. Rubenstein y R. Barnaby	WordStar	1er procesador de textos
1980	Departamento de Defensa EEUU	Ada	Definición del lenguaje Ada
1981	Microsoft	MS-DOS	Desarrollo del Sistema Operativo, adoptado para el IBM-PC
1981	IBM	IBM-PC	Introducción en agosto de un computador personal con una arquitectura "abierta". Utiliza el PC-DOS
1983	B. Stroutstrup (AT & T Bell Labs.)	C++	Ampliación del lenguaje C, orientada a objetos
1984		MIDI	Desarrollo de MIDI (interfaz digital para instrumentos de música)
1984	Sony y Philips	CD-ROM	Gran aceptación de un nuevo sistema para almacenar datos digitales
1985	Thinking Machines y Cray Research	Connection Machina, y Cray-2	Dos supercomputadores que alcanzan tasas de 1000 MIPS
1985	Intel	Intel 80386	1er procesador de 32 bits, incluyendo es su interior la unidad de gestión de memoria
1986	Cray Research	Cray XP	Supercomputador con 4 procesadores y 713 Mflops
1989-1990	T. Bernes-Lee (CERN)	World Wide Web	Propuesta de la web, y diseño de un prototipo utilizando URLs, HTML y http
1991	Cray Research	Cray Y-MP C90	Supercomputador con 16 procesadores y tasas de 16 Gflops
1991	IBM, Motorota y Apple	PowerPC	Anuncio de una nueva familia de microprocesadores (30 de julio)
1992	DEC	Microprocesador Alfa	1er microprocesador (RISC) de 64 bits
1993	Apple	Newton	1er PDA ampliamente usada, con reconocimento de caracteres escritos manualmente
1994	Netscape	Navegador Netscape	1er buscador en web (septiembre)
1995		Java	Lenguaje de programación, independiente de la plataforma, para desarrollar aplicaciones

20.8 Perspectiva actual

La informática es una rama de la ciencia y de la tecnología que en poco más de 50 años de historia ha evolucionado vertiginosamente, siendo difícil predecir incluso el próximo futuro. Como ejemplo del riesgo (en este caso de *quedarse cortos*) que supone realizar predicciones podemos repetir el texto que apareció en la revista americana *Popular Mechanis* en Marzo de 1949, refiriéndose al ENIAC: *Mientras el ENIAC está equipado con 18.000 válvulas de vacío y pesa 30 toneladas, los computadores del futuro pueden tener 1.000 válvulas y tal vez pesen tan sólo una tonelada y media* [Pat00]. Con la ventaja que tenemos actualmente de disponer de cierta perspectiva (que no se tenía en 1949), a continuación comentamos brevemente algunos aspectos relacionados con el momento actual y el fututo de la informática.

20.8.1 Límites de la tecnología: computadores cuánticos y moleculares

Las mejoras introducidas en la tecnología electrónica han hecho posibles los avances en cuanto a incremento de velocidad y a miniaturización de los sistemas informáticos. Hace unos 50 años Gordon Moore, que fue uno de los fundadores de Intel, predijo que el número de transistores que podían ser integrados en un chip crecería exponencialmente, duplicándose aproximadamente cada año y medio. Esta predicción se conoce como **ley de Moore** y se ha verificado hasta la fecha muy razonablemente. El incremento en la miniaturización implica también un aumento en la velocidad, al tener que recorrer las señales distancias más pequeñas. De acuerdo con esta ley para el año 2010 las dimensiones que tendrían que tener los transistores serían tan pequeñas que no se cumplirán las leyes de la física clásica, y habría que revisar los modelos actuales para incluir fenómenos de la física cuántica, La *Semiconductor Industry Association* ha previsto que para finales del año 2018 se dispondrá de circuitos funcionando a 53,2 GHz conteniendo hasta unos 14 mil millones de transistores; y en el caso de microprocesadores se obtendrían densidades de integración del orden de 3.500 millones de transistores por centímetro cuadrado [Fly05].

Constantemente se están investigando y desarrollando nuevos dispositivos electrónicos que tratan de mantener la mejora creciente de la tecnología electrónica que da soporte a los computadores. Así, por ejemplo, la Universidad de Illinois en el momento de redactar estas líneas ha presentado un prototipo de transistor láser que puede realizar setecientos diez mil millones de conmutaciones por segundo (710 GHz), siendo en la actualidad el transistor más rápido del mundo. Los materiales que utiliza son Fosfito de Indio y Arseniuro de Galio, los mismos semiconductores que tradicionalmente se utilizan en la construcción de diodos láser y diodos emisores de luz. Este transistor a partir de un estímulo eléctrico de entrada genera a su salida tanto una señal eléctrica como una señal óptica (haz láser). El haz láser puede ser directamente modulado por el propio transistor pudiendo así transmitir señales con unas tasas de datos superiores a 100 Gbps. Más detalles sobre estos transistores pueden encontrarse en la referencia [Hol06].

Para soslayar las limitaciones en el desarrollo de la microelectrónica, se están investigando dos alternativas: la **computación cuántica** y la **computación molecular**. La mecánica cuántica permite codificar la información en bits cuánticos denominados **qubits**, que pueden almacenar simultáneamente tanto un 0 como un 1. Además se pueden realizar con qubits las operaciones básicas del álgebra de conmutación, con lo que en principio sería factible desarrollar **computadores cuánticos** [Ste00]. La otra alternativa, conocida como **nanotecnología molecular**

consiste en construir materiales y dispositivos átomo a átomo (*de abajo a arriba*) ubicando cada uno en la posición deseada. El objetivo es construir sistemas de **máquinas moleculares**, de manera semejante a como la naturaleza forma los diferentes sistemas: molécula a molécula. En este sentido IBM ha desarrollado un microscopio de tipo barrido/túnel que permite ver y desplazar átomos aislados. Con este sistema un grupo de físicos ha creado un diminuto conmutador que da paso a un diminuto átomo. Desde otro punto de vista, grupos de biofísicos han logrado realizar un rotor proteínico que hace girar de forma controlada la cola de una bacteria flagelada, a partir del cual esperan desarrollar un motor molecular y toda una serie de máquinas moleculares naturales [Bee01]. Este campo es eminentemente interdisciplinar (química, física, ingeniería mecánica, ciencia de materiales, biología molecular e informática), y trabajan en él organizaciones tan relevantes como la NASA, IBM y Caltech [Pet00]. Uno de los retos más ambiciosos de la computación cuántica y molecular se encuentra en la realización de interfaces con los sistemas electrónicos actuales.

20.8.2 Importancia creciente del paralelismo. Supercomputadores

Como se ha puesto de manifiesto a lo largo de este texto (véase, por ejemplo, Sección 1.3) las prestaciones de un computador vienen determinadas por la tecnología y por la arquitectura. Con frecuencia la tecnología por sí sola no proporciona prestaciones (velocidad) suficiente para atender problemas reales de cálculo intensivo. La arquitectura de computadores tiene por objeto diseñar sistemas que satisfagan necesidades del mercado utilizando las posibilidades que ofrece la tecnología; o, en otras palabras, traducir las aptitudes que ofrece ésta última a prestaciones reales. Cuando cambian las demandas del mercado los arquitectos de computadores deben ingeniárselas para satisfacerlas a un costo razonable, con la tecnología existente en ese momento. Para lograr ese valor añadido la principal opción encontrada ha sido la utilización del concepto de paralelismo, que permite, a distintos niveles, reducir el tiempo gracias a la realización simultánea de operaciones (Sección 12.7).

Como ejemplo de proyecto actual se puede citar el acometido por IBM, Sony y Toshiba, con un presupuesto de 400 millones de dólares y en el que han llegado a trabajar 400 personas, para realizar un multiprocesador en un chip que será utilizado en la consola del la PlayStation 3 de Sony y en sistemas de TV. Este chip contiene 9 procesadores (núcleos o *cores*), uno de ellos similar a un PowerPC y los otros 8 están diseñados para aplicaciones de gráficos y de multimedia de banda ancha [Moo06]. En cuanto a multiprocesadores en un chip de uso general se dispone de varios modelos comercializados que incluyen 8 procesadores, como es el caso de Sun Niagara e IBM Power5.

Especial importancia tiene la utilización de paralelismo para la construcción de supercomputadores, en un grado que Gordon Bell denomina *petaescala* [Bel06]. Un computador de petaescala estará caracterizado por realizar petaoperaciones por segundo y utilizar memoria masiva del orden de petabytes[1].

Como precedentes de *computadores petaescala* podemos citar a los sistemas de comunicación y acceso a la información a través de Internet de Google, Yahoo!, AOL y MSN. Estos buscadores están constituidos por computadores paralelos que disponen de decenas de miles de nodos de procesamiento (con una capacidad de cerca de 1 Petaflops) y tienen cerca de cien mil unidades locales de disco para satisfacer los requisitos de ancho de banda. Se necesitan aún mayores capacidades de computación para procesar información a nivel semántico, cosa que cada vez es más requerida por los usuarios de los sistemas de búsqueda citados.

[1] Recuérdese, Apéndice B, que 1 peta = 10^{15}; es decir 1 petaoperaciones son 1.000 billones de operaciones por segundo. Por otra parte, 1 Petabyte = 2^{50} Bytes ≈ mil millones de Megabytes.

El ámbito tradicional de utilización de los supercomputadores es el de cálculo intensivo, haciendo que una de las características más sobresalientes de la informática sea posibilitar notablemente el avance de prácticamente la totalidad de las ramas de las ciencias. Como indica Duato en [Dua05], se puede afirmar que ciencias que habían estado medio dormidas durante siglos han sufrido de repente un nuevo despertar y una rápida evolución. Tal es el caso de la Medicina y la Biología con sus últimos avances en el campo de la Genética, siendo uno de los ejemplos recientes la secuenciación del cromosoma de la especie humana, cosa que hubiera requerido siglos de experimentación de no ser por la posibilidad de hacer búsquedas rápidas con ordenador de cadenas en el genoma de un determinado organismo y comparaciones con las mismas. Estas búsquedas requieren enormes bases de datos y supercomputadores paralelos con un gran número de procesadores para poder hacerlas en un tiempo razonable. Muchos de los complejos cálculos que ahora se hacen de forma cotidiana, tales como el cálculo de cascos de veleros para competición, el cálculo de la aerodinámica de los aviones, o los cálculos necesarios para generar el movimiento de los personajes en películas de animación, no serían posibles si no dispusiéramos de potentes supercomputadores paralelos.

Tradicionalmente se ha considerado que las ciencias experimentales tienen dos grandes ramas: *teoría* y *experimentación*; con el paso del tiempo ha ido adquiriendo tal importancia la informática que se puede afirmar que ha surgido otra rama nueva que podríamos denominar *computacional*, cuyo objetivo es encontrar soluciones a modelos matemáticos complejos [Gor06].

20.8.3 Accesibilidad a recursos informáticos y a la información

Un aspecto fundamental del la informática actual es el incremento descomunal que se ha producido en cuanto a la facilidad de accesibilidad a los recursos informáticos, y en particular a la información.

En diversas partes de este texto (véase, por ejemplo, la Sección 1.5 y 12.3) hemos indicado que los procesadores no sólo se utilizan en computadores de uso general, sino también embebidos en multitud de sistemas. Cada vez se están introduciendo más este tipo de sistemas tanto en la electrónica de consumo como en otras actividades (redes de datos, industria del automóvil, edificios inteligentes, domótica, etc.). Podríamos redefinir las cuatro generaciones, en que tradicionalmente se considera dividida la historia de la Informática descritas de la Sección 20.2 a 20.5 del presente capítulo, atendiendo a la disponibilidad de acceso a los computadores, según se indica en la Tabla 20.6.

Tabla 20.6. *Eras de computadores según su disponibilidad.*

• Grandes computadores (uno por empresa); 1950
• Minicomputadores (uno por departamento); 1970
• PCs (uno por persona); 1980
• Embebidos (varios por persona, en múltiples sistemas): 2000

En otro orden de cosas, además de las tradicionales redes de comunicaciones de área amplia, área metropolitana y área local, ha surgido el concepto de **redes de área personal** (Sección 19.1.4), que permite, por ejemplo, intercomunicar sin utilización de ningún tipo de cables los distintos periféricos de un computador o incluso distintos dispositivos de comunicaciones (teléfonos móviles, por ejemplo) y equipos de electrónica de consumo (sistemas de audio, TV, etc.), todos ellos entre sí. Esto está siendo posible gracias a la utilización de redes WiFi, Bluetooth o

de infrarrojos. En un futuro no muy lejano es previsible que cualquier aparato electrodoméstico pueda estar conectado a nuestro PC, incluyendo el calentador de agua, la lavadora, el lavaplatos o el sistema de depuración del agua de la piscina. Ya se está previendo que en las instalaciones de viviendas (incluso particulares) se incluya una infraestructura de red de comunicaciones (bien bajo la forma de fibra óptica o de red inalámbrica) de igual forma a como se dispone de una infraestructura de cableado para suministro de la energía eléctrica y de fontanería.

Es claro que poco a poco las comunicaciones inalámbricas están sustituyendo a las tradicionales cableadas; pudiendo citar como ejemplos las redes BeeZep, para edificios inteligentes, y WiMax, inalámbrica tipo WiFi con un alcance superior a 45 Km. Estas últimas redes se han empezado a utilizar por proveedores de Internet que no disponen de una red pública propia de transmisión de datos o a las que no pueden conectarse sus clientes; éste es el caso de un proyecto que se está desarrollando en Malasia que trata de proporcionar una comunicación inalámbrica de banda ancha que proporciona a los usuarios un acceso equivalente a DSL (1 a 3 Mbps) que pueda transmitir también llamadas de teléfono de alta calidad [Che06].

También se han desarrollado microsensores miniaturizados y de muy bajo consumo (con una pequeña pila pueden tener una autonomía de hasta 3 ó 4 años) que incluyen la interfaz y protocolo de comunicaciones con tecnología WiFi, WiMax o ZigBee. Con este tipo de sensores se puede establecer, por ejemplo, una red en un bosque y visualizar a través de internet las condiciones ambientales de sus distintos puntos con objeto de detectar un foco de incendio o el nivel de riesgo de que se produzcan éstos.

Poco a poco las redes se van imbricando más con la arquitectura de computadores, de forma tal que está incrementándose notablemente el solapamiento entre *procesamiento* de datos y *comunicaciones* de datos, dando lugar a una mayor integración entre los sistemas que *procesan* información y los que la *transmiten*. Las aplicaciones a través de Internet están incluyendo cada vez más información multimedia bajo diversas formas de secuencias de vídeo, servicios en tiempo real, tales como telefonía por Internet y servicios asociados de movilidad, y diferentes formas de comercio electrónico. Las redes deben proporcionar un ancho de banda adecuado para admitir este tipo de aplicaciones; para lo cual en la actualidad se pretende introducir nuevos dispositivos ópticos, siendo previsible disponer en un futuro de conductos ópticos, formados por cientos de fibras, que transmitan en el rango de petabits por segundo (10^{15} bps) [Leo01, Rie01].

A pesar de que el objetivo de las redes es dar soporte a aplicaciones, realizándose el procesamiento de éstas en los puntos terminales de la red (donde se ubican los servidores y los clientes), funciones tradicionalmente realizadas en éstos se están desplazando a servidores propios de la red bajo la forma de servidores de nombres, servidores de localización para permitir la comunicación con usuarios móviles, cortafuegos, almacenamiento temporal y sistemas de apoyo a multimedia.

Por otra parte se está observado un incremento espectacular en las ventas de distintos tipos de sistemas específicos para conexión a internet (**computadores de red**, o *Network Computers*), en detrimento de los clásicos PC. Ello es debido a que los sistemas PC se proyectaron como sistemas autónomos con grandes recursos propios de memoria. Por lo general un PC conectado a la red no necesita hacer uso de sus numerosas posibilidades; en cierta medida puede decirse que está sobredimensionado, e Internet está dejando obsoleta la concepción del computador como elemento aislado para procesamiento. Los nuevos computadores de red, sin embargo, se proyectan teniendo en cuenta el entorno (red) en el que van a trabajar y que cada vez más la información se encuentra en la red y no en el terminal del usuario.

Hay una importancia creciente en la interconexión de computadores frente a cualquier otro dispositivo de cómputo. Como indica Hennessy [Hen99] está surgiendo un nuevo enfoque en la investigación e industria de los computadores, de forma que, aunque la velocidad seguirá siendo

importante, la clave será la *velocidad de comunicación*, y otras propiedades como la *disponibilidad de aplicaciones y servicios*, la facilidad de *mantenimiento de los sistemas* y su *escalabilidad*.

Según Duato [Dua05] la combinación de los nuevos procesadores (multiprocesadores en un chip) junto las posibilidades que ofrecen las comunicaciones inalámbricas hará posible el diseño de dispositivos electrónicos de bolsillo con funciones que ahora parecen de ciencia-ficción. Por ejemplo, entenderán nuestras órdenes habladas y las obedecerán, podrán comprimir y descomprimir vídeo de alta resolución en tiempo real, generarán, también en tiempo real, imágenes tridimensionales que representarán en el cristal de unas gafas especiales, permitiéndonos así un rápido acceso a la información visual, etc. Con probabilidad un turista podrá solicitar, mientras camina por una ciudad, información detallada sobre el punto donde se encuentra, pudiendo consultar mapas, fotografías y vídeos de dicho lugar. Otro ejemplo más futurista es la utilización de las redes de sensores miniatura, descritos anteriormente, que se podrán desplegar con gran rapidez y que permitirán la monitorización de multitud de eventos de forma remota para, por ejemplo, la localización de francotiradores en zonas de conflicto bélico, hasta llegar a la implantación de una red de sensores en Marte que permita captar de forma continuada datos de diferentes puntos de la superficie del planeta. La inmensa cantidad de información recopilada por estas últimas redes de sensores requerirá para procesarla de las enormes potencias de cálculo citadas en la Sección 20.8.2.

En el campo de los periféricos también se están logrando desarrollos notables. Como ejemplo podemos citar los visualizadores (*displays*) nanocrómicos. Uno de los problemas que se plantea en multitud de sistemas embebidos es que disponen de visualizadores de cristal líquido (LCD, Sección 11.3.2.2), los cuales tienen muy bajo contraste. La empresa irlandesa Ntera [Per06] recientemente ha desarrollado unos nuevos visualizadores basados en el uso de moléculas electrocrómicas que tienen una relación de contraste 6:1 y un brillo de al menos 4 veces la de un LCD, viéndose la imagen desde un gran ángulo e incluso con muy baja luminosidad, recordando la calidad de percepción a la escritura realizada sobre papel. Además el consumo de energía es muy inferior al de los LCD por no necesitar una fuente de luz interna ni refrescar constantemente la información de cada píxel. El coste de fabricación también resulta muy inferior a la de los visualizadores que pretende sustituir. Los materiales electrocrómicos tienen la capacidad de cambiar persistente y reversiblemente sus propiedades ópticas (transmitancia y/o reflectancia) cuando son sometidos a potenciales eléctricos; cuando dicho potencial se suspende o se invierte, estos materiales mantienen o recobran su estado original de forma reversible. Esta variación de las propiedades ópticas se produce debido a la inserción/extracción de cationes en la película electrocrómica.

20.8.4 Nuevas concepciones sobre computación: sistemas bioinspirados

La metodología seguida por las ingenierías y por las ciencias naturales están claramente diferenciadas, siendo el objetivo de las primeras construir sistemas artificiales para satisfacer necesidades concretas y el de las segundas modelar para predecir comportamientos futuros. No obstante, han florecido diversos movimientos que tratan de establecer puentes entre ambas metodologías. Uno de ellos parte de la observación de que los seres vivos son sistemas complejos que exhiben características interesantes tales como evolución, adaptación y tolerancia a fallos, que sin embargo son difíciles de conseguir con la metodología convencional de la ingeniería.

Dentro de este contexto han surgido tres metodologías de especial interés: las redes neuronales artificiales [Hay99], los sistemas difusos [Jan97], y la computación evolutiva [Mic99].

Todas ellas tienen en común que, en cierta medida, tratan de imitar la forma en que el hombre, los animales y otros sistemas biológicos procesan la información, evolucionan y aprenden procedimientos para resolver problemas difíciles de solucionar utilizando los métodos convencionales de la informática. Por ello, a los productos obtenidos usando dichas metodologías se les puede agrupar con la denominación de **sistemas inteligentes** o **sistemas bioinspirados**.

La motivación del nacimiento del campo de las redes neuronales artificiales, que son sistemas que tratan de emular más o menos fidedignamente a las redes neuronales naturales, se encuentra en la observación de que el cerebro procesa la información de forma completamente diferente a como lo hacen los computadores digitales convencionales. El conocimiento moderno del cerebro parte de los trabajos del español Ramón y Cajal, que introdujo la idea de neurona como elemento estructural constitutivo del cerebro. Típicamente las neuronas son cinco órdenes de magnitud más pequeñas que las puertas de silicio, pero son un millón de veces más lentas que estas últimas. Sorprendentemente, a pesar de esta relativa lentitud, el tejido nervioso realiza ciertas computaciones (como reconocimiento de patrones, percepción y control motor) mucho más rápidamente que los computadores digitales actuales. Además, se estima que el cerebro posee una eficiencia energética (medida en julios por operación por segundo) de 10^{10} veces superior a la de los computadores actuales más potentes.

Bajo el punto de vista de la ingeniería de computadores, el cerebro es un sistema de procesamiento de la información, en definitiva un computador paralelo, no lineal y altamente complejo.

Una red neuronal artificial está formada por un conjunto de elementos de cómputo (neuronas), usualmente distribuidas en capas y con un gran número de interconexiones entre ellas. El objetivo del campo de las redes neuronales artificiales es construir sistemas que tengan la posibilidades de aprender y generalizar, siendo capaces de resolver tareas difícilmente resolubles con la computación convencional (simbólica). Es deseable que estos sistemas exhiban características que son comunes en los sistemas biológicos, tales como computación asíncrona masivamente paralela, comportamiento tolerante a fallos, y funcionamiento en tiempo real incluso en entornos complejos y cambiantes.

Los **sistemas difusos** surgen de la observación de que el cerebro biológico tiene la capacidad de procesar información imprecisa o cualitativa, frases como "esta persona no es demasiado joven", "esta ropa está grasienta", "hay excesiva luz" son analizadas perfectamente en nuestro cerebro, y a pesar de su imprecisión nos permiten tomar decisiones acertadas. Este tipo de expresiones, que definen conceptos en lenguaje natural, se denominan **variables lingüísticas** y, claramente, las técnicas informáticas clásicas no proporcionan herramientas efectivas para su tratamiento, a pesar de ser utilizadas constantemente por el ser humano.

Para poder procesar el léxico impreciso, L. A. Zadeh ideó la **teoría de conjuntos difusos** [Dub80, Jan97], con ayuda de la cual se pueden construir sistemas difusos (y en particular controladores difusos) que, en general para numerosas aplicaciones, resultan menos complejos y costosos que los sistemas tradicionales, debido a que estos últimos suelen utilizar una precisión y recursos de cómputo innecesarios.

Dentro del campo de la computación evolutiva, caben destacar los algoritmos genéticos, que se aplican en la resolución de problemas de optimización, reconocimiento y búsqueda. Parten de la observación de que los animales y las plantas, en sucesivas generaciones, han sido capaces de adaptarse a cambios en su forma de vida, e incluso en su estructura, para combatir cualquier agente que hiciera peligrar su supervivencia (evolución de Darwin). Con los **algoritmos genéticos** se trata de emular a la naturaleza: se asume por ésta aquello que sea favorable para la supervivencia de la población, mientras que aquello que represente debilidad (dificultad en la búsqueda de comida, poca adaptación al entorno –sequía–, etc.) se descarta.

Las metodologías bioinspiradas citadas anteriormente, usadas aisladamente o en combinación (sistemas híbridos), permiten abordar nuevas aplicaciones en campos tales como la teoría de control, robótica, procesamiento de señal, gestión administrativa, predicción temporal, planificación financiera y seguridad, e incluso en el propio campo de la ingeniería de computadores.

Una aplicación de los sistemas bioinspirados relacionada directamente con la tecnología de computadores es el campo de las **arquitecturas evolutivas** y **autorreparadoras**, en las que los circuitos se autoconfiguran para realizar una función definida (como la información del ADN en los sistemas biológicos). Si uno de los módulos de estos sistemas sufre un funcionamiento anómalo o una rotura, el sistema se reconfigura automáticamente para seguir realizando su función, aunque sea en forma degradada.

20.9 Conclusión

Hemos presentado en este capítulo una breve descripción de la evolución de la apasionante y cambiante ciencia y tecnología de la informática, que a pesar de contar con poco más de medio siglo de vida, ha puesto en manos de la sociedad recursos que posibilitan el avance de prácticamente la totalidad de otras ciencias y que el hombre se dedique a tareas más superiores y creativas. También hemos incluido al final del capítulo una sección que da idea de algunas de las líneas actuales de avance de la informática.

EJERCICIOS

20.1 Realizar la multiplicación siguiente 3.547 x 163 utilizando números romanos.

20.2 Aplicando los principios de la Máquina de Diferencias de Babage, utilizando un desarrollo en serie, y aproximando por los dos primeros términos, obtener una tabla de los valores de sen(x), para x = 0,0; 0,1; 0,2; 0,3; 0,4; 0,5; etc.

20.3 Realice con relés los siguientes circuitos lógicos: AND, OR, e INVERSOR. *Sugerencia:* las entradas se realizan a través de las bobinas de excitación. Considerar la existencia de dos tipos de relés: uno como el de la Figura 20.5, que cuando se aplica 0 el circuito permanece abierto, y cuando se aplica 1 se cierra, y otro que funciona inversamente; es decir, cuando la entrada es 0 permanece cerrado, y cuando es 1 se abre.

BIBLIOGRAFÍA

[Bek01] Beekman, G.: *Computer Confluence. Exploring Tomorrow's Technology*, 4ª Edc. Prentice Hall, 2001.

[Bel06] Bell, G.; Gray, J.: *Petascale Computacional Systems*, IEEE Computer, Vol. 39, N0.1, pp. 110-112, Enero 2006.

[Car96] Carlson, B.; Burgués, A.; Millar, C.: *Timeline of Computing History*, IEEE, 1996. http://www.computer.org/portal/cms_docs_computer/computer/timeline/timeline.pdf.

[Che06] Cherry, S.: *A radio bypass*, IEEE Spectrum, Vol. 43, Nº 1, pp. 30-33, Enero, 2006.

[Dua05] Duato, J. *Presentación del texto;* Ortega, J.; Anguita, M.; Prieto, A.: *Arquitectura de Computadores*, pp. xv-xvii, Thomson, 2005.

[Dub80] Dubois, D.; Prade, H.: *Fuzzy Set and Systems: Theory and Applications;* Academic Press, 1980.

[Ell01] Elliott, C.: *Building the Wireless Internet*; IEEE Spectrum, Vol. 38, Nº 1, pp. 14-16, Enero 2001.

[Fly05] Flynn, M. J.: *Microprocessor Design Issues: thoughts on the road ahead,* IEEE Micro, Vol. 25, No. 23, pp.16-31, Mayo-Junio, 2005.

[Hay79] Hayes, J. P.: *Computer Architecture and Organization.* McGraw-Hill, 1979.

[Hay99] Haykin, S.: *Neural Networks. A comprehensive Foundation;* 2ª Edic.; Prentice Hall, 1999.

[Hen99] Hennessy, J. L.: *The Future of System Research*; IEEE Computer, pp. 27-33, Agosto 1999.

[Hol06] Holonyak, N.; Fena, M.: *The Transistor Laser*, IEEE Spectrum, Vol. 43, Nº 2, pp. 150-55, Febrero 2006.

[Hux76] Huskey, H. D.; Huskey, V. R.: *Chronology of Computing Devices.* IEEE Transactions on Computers, Vol. C-25, Nº 12, pp. 1190-1199, Dec. 1976.

[Jan97] Jang, J. S. R.; Sun, C. T.; Mizutani, E.: *Neuro-fuzzy and Soft Computing,* Prentice Hall, 1997.

[Leo01] León-García, A.; Widjaja, I.: *Redes de comunicación. Conceptos fundamentales y arquitecturas básicas;* MacGraw-Hill, 2001.

[Ort05] Ortega, J.; Anguita, M.; Prieto, A.: *Arquitectura de Computadores*, Thomson, 2005.

[Pat00] Patterson, D. A., Hennessy, J. L.: *Estructura y diseño de computadores;* Editorial Reverté, 2000.

[Per06] Perry, T. S.: *Black, white, and readable*, IEEE Spectrum, Vol. 43, No. 1, pp. 36-39, Enero, 2006.

[Pet00] Peterson, C.: *Taking Technology to the Molecular Level,* IEEE Computer, Vol. 33, Nº 1, págs.46-53, Enero 2000.

[Ran70] Randell, B. (Editor): *The origins of Digital Computers.* Springer-Verlag. 1970.

[Rie01] Riezenman, M. J.: *Optical Nets Brace for Even Heavier Traffic;* IEEE Spectrum, Vol. 38, Nº 1, págs. 44-46, Enero 2001.

[Ste00] Steane, A. M.; Rieffel, E. G.: *Beyond biys: Quantum Information Proccesing;* Computer, IEEE, Vol. 33, Nº1, págs.38-45, Enero 2000.

[Wex81] Wexelblat, R. L. (Editor): *History of Programming Languages.* ACM Monograph Series, Academic Press, 1981.

[Wil85] Williams, M. R.: *A History of Computing Technology*. Prentice-Hall, 1985.

APÉNDICE A

Sistemas de numeración usuales en informática

Los computadores suelen efectuar las operaciones aritméticas utilizando una representación para los datos numéricos basada en el **sistema de numeración base dos** (que, por abreviar, denominaremos **binario natural,** o **binario,** sin más).

También se utilizan los sistemas de numeración octal y hexadecimal, para obtener códigos intermedios. Un número expresado en uno de estos dos códigos puede transformarse directa y fácilmente a binario y viceversa. Con ellos se simplifica la trascripción de números binarios y se está más próximo al sistema que utilizamos usualmente (el sistema decimal), por lo que con gran frecuencia se utilizan como paso intermedio en las transformaciones de decimal a binario y viceversa. Además la realización electrónica de codificadores/decodificadores entre binario y un código intermedio es mucho más simple que entre binario y decimal.

El objetivo de este apéndice es resumir los aspectos prácticos de mayor interés de los sistemas de numeración desde el punto de vista de su uso en los computadores.

A.1 Representación posicional de los números

Un **sistema de numeración en base *b*** utiliza para representar los números un alfabeto compuesto por *b* símbolos o cifras. Así, todo número se expresa mediante un conjunto de cifras, contribuyendo cada una de ellas con un valor que depende de:

a) la cifra en sí, y

b) la posición que ocupe dentro del número.

En el **sistema de numeración decimal** (o **sistema en base 10**), que es el que habitualmente se utiliza, $b = 10$ y el alfabeto está constituido por diez símbolos, denominados también cifras decimales:

[A.1]

$$S_{10} = \{0,1,2,3,4,5,6,7,8,9\}$$

EJEMPLO A.1

El número decimal 3.278,52 puede obtenerse como suma de:

$$\begin{array}{r} 3000 \\ 200 \\ 70 \\ 8 \\ 0{,}5 \\ 0{,}02 \\ \hline 3.278{,}52 \end{array}$$

es decir, se verifica:

$$3.278{,}52 = 3 \cdot 10^{3} + 2 \cdot 10^{2} + 7 \cdot 10^{1} + 8 \cdot 10^{0} + 5 \cdot 10^{-1} + 2 \cdot 10^{-2}$$

Cada posición tiene un nombre y peso específicos:

posición 0 peso $b^0 = 1$, unidades (en el ejemplo: 8)
posición 1 peso $b^1 = 10$, decenas (en el ejemplo: 7)
posición 2 peso $b^2 = 100$, centenas (en el ejemplo: 2)
posición 3 peso $b^3 = 1.000$, millares (en el ejemplo: 3)
etc.
posición –1 peso $b^{-1} = 0{,}5$, décimas (en el ejemplo: 5)
posición –2 peso $b^{-2} = 0{,}225$, centésimas (en el ejemplo: 2)
etc.

Generalizando para cualquier base, se tiene que la representación de un número N en la base b:

[A.2]

$$N \equiv \dots n_4 n_3 n_2 n_1 n_0 n_{-1} n_{-2} n_{-3} n_{-4} \dots$$

es una forma abreviada de expresar su valor, que es:

[A.3]

$$N \equiv \dots n_4 \cdot b^4 + n_3 \cdot b^3 + n_2 \cdot b^2 + n_1 \cdot b^1 + n_0 \cdot b^0 + n_{-1} \cdot b^{-1} + n_{-2} \cdot b^{-2} + n_{-3} \cdot b^{-3} + n_{-4} \cdot b^{-4} + \dots$$

Para representar una cantidad, por un lado resulta más cómodo que el número de símbolos (cifras) del alfabeto (la base) sea lo menor posible, pero, por otra parte, cuanto menor es la base, mayor es el número de cifras que se necesitan para representar una cantidad dada.

En la Tabla A.1 se definen los sistemas de numeración usuales en informática; estos son: el decimal, binario, octal y hexadecimal. Los dos últimos sistemas se conocen con el nombre de códigos intermedios, ya que con mucha frecuencia no se transforma un número directamente de decimal a binario, sino que se hace a través de octal o hexadecimal como se verá en la Sección A.4.

Tabla A.1. *Sistemas de numeración usuales en informática.*

Denominación	base	Símbolos utilizados
Decimal	b = 10	$S_{10} = \{0,1,2,3,4,5,6,7,8,9\}$
Binario	b = 2	$S_2 = \{0,1\}$
Octal	b = 8	$S_8 = \{0,1,2,3,4,5,6,7\}$
Hexadecimal	b = 16	$S_{10} = \{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F\}$

A.2. Operaciones aritméticas con variables binarias

Las operaciones aritméticas básicas son la suma, resta, multiplicación y división. Al ser la representación en binario natural una notación ponderada, es decir, verificar la expresión [A.3], estas operaciones son análogas a las realizadas en decimal pero hay que utilizar para cada posición las Tablas A.2, A.3, A.4, y A.5, según la operación aritmética a realizar. A continuación se incluyen unos ejemplos.

Tabla A.2. *Suma aritmética con variables binarias.*

a	b	$a + b$
0	0	0
0	1	1
1	0	1
1	1	0 *y me llevo* 1

Tabla A.3. *Resta aritmética con variables binarias.*

a	b	$a - b$
0	0	0
0	1	1 *y adeudo* 1
1	0	1
1	1	0

Tabla A.4. *Multiplicación aritmética con variables binarias.*

a	b	$a \cdot b$
0	0	0
0	1	0
1	0	0
1	1	1

Tabla A.5. *División aritmética con variables binarias.*

a	b	a/b
0	0	*Indeterminado*
0	1	0
1	0	∞
1	1	1

EJEMPLO A.2

Efectuar las siguientes operaciones aritméticas binarias:

1110101 + 1110110; 1101010 – 010111;
1101010 x 11; 1010011 x 10 y 1101,01/101

SOLUCIÓN

$$\begin{array}{r} 1110101 \\ +\;1110110 \\ \hline 11101011 \end{array} \qquad \begin{array}{r} 1101010 \\ -\;1010111 \\ \hline 0010011 \end{array} \qquad \begin{array}{r} 1101010 \\ \times\quad 11 \\ \hline 1101010 \\ 1101010\; \\ \hline 100111110 \end{array} \qquad \begin{array}{r} 1010011 \\ \times\quad 10 \\ \hline 0000000 \\ 1010011\; \\ \hline 10100110 \end{array}$$

$$\frac{1101{,}01}{101} = \frac{110101}{10100}$$

$$\begin{array}{r|l} 110101 & 10100 \\ \cline{2-2} -10100 & 10{,}101\ldots \\ \cline{1-1} 0011\mathbf{\mathit{10}} & \\ -10100 & \\ \cline{1-1} 0011\mathbf{\mathit{00}} & \\ -10100 & \\ \cline{1-1} 00010 & \end{array}$$

Se observa que multiplicar por $10)_2$ (es decir, por 2 en decimal) equivale a añadir un cero a la derecha, siendo esto similar a multiplicar por $10)_{10}$ un número decimal. De la misma forma dividir por $2)_{10} = 10)_2$ se hace desplazando el punto decimal a la izquierda, o eliminando ceros a la derecha.

EJEMPLO A.3

Se verifican las siguientes igualdades

$$10101{,}0101)_2 \cdot 2 = 101010{,}101)_2$$
$$1101{,}1010)_2 \cdot 2^5 = 110110100)_2$$
$$1010100)_2/2 = 101010)_2$$
$$10101{,}101)_2/2^{-6} = 0{,}010101101)_2$$

A.3. Representación en complementos

Para representar un número negativo se puede utilizar el complemento de ese número a la base o a la base menos uno del sistema de numeración utilizado. De esta forma, como se verá más adelante, las sumas y restas quedan reducidas a sumas, independientemente de los

signos de los operandos. Este sistema de representación es de sumo interés en el caso de los computadores ya que al utilizarlo se reduce la complejidad de los circuitos de la unidad aritmético-lógica (no son necesarios circuitos específicos para restar, Sección 9.1.2.1).

El **complemento a uno** de un número binario, *N*, es el número que resulta de sustituir unos por ceros y ceros por unos en el número original.

Podemos **restar dos números binarios** sumando al minuendo el complemento a uno del sustraendo. La cifra que se arrastra del resultado se descarta y se suma al resultado así obtenido.

EJEMPLO A.4

El complemento a uno del número 10010 es 01101.
El complemento a uno de 101010 es 010101.
El resultado de restar 1000111 – 10010 es:

$$\begin{array}{r} 1000111 \\ -\ 0010010 \\ \hline \mathbf{0110101} \end{array}$$

o bien, como el complemento a 1 de 0010010 es 1101101

$$\begin{array}{r} 1000111 \\ +\ 1101101 \\ \hline (1)0110100 \\ \rightarrow +\ 1 \\ \hline \mathbf{0110101} \end{array}$$

El **complemento a dos** de un número binario, *N,* es el número que resulta de sumar uno al complemento a uno de dicho número.

Se pueden restar dos números binarios sumando al minuendo el complemento a dos del sustraendo y despreciando, en su caso, el acarreo del resultado.

EJEMPLO A.5

El complemento a 2 del número 10010 es 01110; en efecto:

$$C_2(10010) = C_1(10010) + 1 = 01101 + 1 = 01110$$

El complemento a 2 del número 101010 es 010110; en efecto:

$$C_2(101010) = C_1(101010) + 1 = 010101 + 1 = 010110$$

El resultado de las restas: 11001 – 10010 y 110000 – 101010 son:

$$\begin{array}{r} 11001 \\ -\ 10010 \\ \hline \mathbf{00111} \end{array} \qquad \begin{array}{r} 110000 \\ -\ 101010 \\ \hline \mathbf{001110} \end{array}$$

O sumando al minuendo el complemento a dos del sustraendo:

$$\begin{array}{r} 11001 \\ +\ 01110 \\ \hline (\not{1})\ \mathbf{00111} \end{array} \qquad \begin{array}{r} 110000 \\ +\ 010110 \\ \hline (\not{1})\ \mathbf{001110} \end{array}$$

A.4. Transformaciones entre bases distintas

Como se verá a continuación, las transformaciones de base decimal a binaria, y viceversa, son relativamente complejas y no se pueden hacer directamente en paralelo en un solo paso, lo cual quiere decir que la realización de estas conversiones tanto por medio de un programa como por circuitos especiales son lentas. No ocurre lo mismo con las conversiones entre octal y binario y entre hexadecimal y binario, por lo que con mucha frecuencia se utilizan estos sistemas de numeración. Es más, con frecuencia para pasar de decimal a binario, y viceversa se suele realizar utilizando como pasos intermedios la base octal o la hexadecimal, en la forma que se indica a continuación:

decimal → hexadecimal → binario
binario → hexadecimal → decimal

decimal → octal → binario
binario → octal → decimal

El código octal se suele utilizar cuando el número de bits a representar es múltiplo de 3, y el hexadecimal cuando dicho número es múltiplo de 4.

A.4.1 Transformaciones con la base decimal

Podemos transformar un número binario a decimal sin más que tener en cuenta las expresiones [A.2] y [A.3], en las que en este caso será $b = 2$, según se pone de manifiesto en los siguientes ejemplos:

EJEMPLO A.6

Transformar a decimal los siguientes números binarios:

110100; 0,10100; 10100,001

SOLUCIÓN

$$110100)_2 = (1 \cdot 2^5) + (1 \cdot 2^4) + (1 \cdot 2^2) = 2^5 + 2^4 + 2^2 = 32 + 16 + 4 = 52)_{10}$$
$$0,10100)_2 = 2^{-1} + 2^{-3} = (1/2) + (1/8) = 0,625))_{10}$$
$$10100,001)_2 = 2^4 + 2^2 + 2^{-3} = 16 + 4 + (1/8) = 20,125)_{10}$$

Observando el Ejemplo A.6 se deduce que podemos transformar de binario a decimal sencillamente sumando los pesos de las posiciones en las que hay un 1, como se muestra en el siguiente ejemplo.

EJEMPLO A.7

Obtener los valores decimales correspondientes a los números binarios 1001,001 y 1001101 número binario

	número binario	*equivalente decimal*
	1 0 0 1, 0 0 1	$= 8 + 1 + 1 + 1/8 = 9,125)_{10}$
pesos →	*8 4 2 1, ½ ¼ 1/8*	

	número binario	*equivalente decimal*
	1 0 0 1 1 0 1	$= 64 + 8 + 4 + 1 = 77)_{10}$
pesos →	*64 32 16 8 4 2 1*	

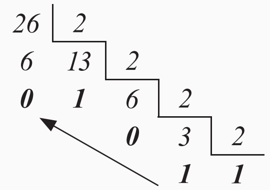
Analicemos ahora como se hace la transformación inversa; es decir, la conversión de un número decimal a binario:

a) La parte entera del nuevo número (binario) se obtiene dividiendo por 2 (sin obtener decimales en el cociente) la parte entera del número decimal de partida, y los cocientes que sucesivamente se vayan obteniendo. Los residuos (restos) de estas divisiones y el último cociente (que serán siempre ceros o unos) son las cifras binarias. El último cociente será el **bit más significativo (MSB)** (*Most Significative Bit*) y el primer residuo será el **bit menos significativo (LSB)** (*Least Significative Bit*).

EJEMPLO A.8

Obtener el equivalente binario del número decimal 26.

$26)_{10}$ *en binario es:* $26)_{10} = 1\ 1\ 0\ 1\ 0)_2$, *ya que:*



Con lo que: $26)_{10} = 1\ 1\ 0\ 1\ 0)_2$

b) La parte fraccionaria del número binario se obtiene multiplicando por 2 sucesivamente la parte fraccionaria del número decimal de partida y las partes fraccionarias que se van obteniendo en los productos sucesivos. El número binario se forma con las partes enteras (que serán ceros o unos) de los productos obtenidos, como se hace en el siguiente ejemplo.

EJEMPLO A.9

Transformar a binario natural el número decimal 0,1875

0,1875	0,375	0,7500	0,5000
x 2	x 2	x 2	x 2
***0**,37500*	***0**,7500*	***1**,5000*	***1**,0000*

Luego: $0{,}1875)_{10} = 0{,}0011)_2$

En consecuencia, el número decimal 26,1875 en binario es:

$26{,}1875)_{10} = 11010{,}0011)_2$

En el último ejercicio del Ejemplo A10 se observa que un número decimal con cifras fraccionarias puede dar lugar a un número binario con un número de cifras fraccionarias mucho mayor o incluso infinito. Si el número binario se almacena con un número prefijado de bits se producirá en la representación binaria un error de truncamiento.

EJEMPLO A.10

Transformar a número binario el número decimal 74,423

a) parte entera:

74 | 2						
14	37 | 2					
	17	18 | 2				
	1	***0***	9 | 2			
			1	4 | 2		
				0	2 | 2	
					0	***1***

b) parte fraccionaria:

0,423	*0,846*	*0,692*	*0,384*	*0,768*
x 2	*x 2*	*x 2*	*x 2*	*x 2*
***0**,846*	***1**,692*	***1**,384*	***0**,768*	***1**,536*

Es decir:

$$74{,}423)_{10} = 1001010{,}01101...)_2$$

En el último ejercicio del Ejemplo A10 se observa que un número decimal con cifras fraccionarias puede dar lugar a un número binario con un número de cifras fraccionarias mucho mayor o incluso infinito. Si el número binario se almacena con un número prefijado de bits se producirá en la representación binaria un error de truncamiento.

A.4.2 Transformaciones con la base octal

Para facilitar la comprensión de este apartado en la Tabla A.6 incluimos las 8 primeras cifras decimales, que coinciden con las cifras octales, y sus correspondientes valores binarios.

Tabla A.6. *Números decimales del 0 al 7 y su correspondiente binario.*

Número decimal	Número binario
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Un número octal puede pasarse a binario aplicando algoritmos análogos a los vistos en la Sección A.4.1. No obstante, al ser $b = 8 = 2^3$, puede hacerse la conversión fácilmente en la forma que se indica a continuación.

Para **transformar un número binario a octal** se forman grupos de tres cifras binarias a partir del punto decimal hacia la izquierda y hacia la derecha. Posteriormente se efectúa directamente la conversión a octal de cada grupo individual.

Como en la práctica sólo es necesario hacer la conversión a octal (o decimal) de cada uno de los grupos de tres cifras, basta con memorizar la Tabla A.6 para poder realizar rápidamente la conversión.

De octal a binario se pasa sin más que convertir individualmente a binario (tres bits) cada cifra octal, manteniendo el orden del número original.

EJEMPLO A.11

Número binario	→	10	001	101	100	,	110	10)$_2$	= 2.154,64)$_8$
Número octal	→	2	1	5	4	,	6	4	

Número binario	→	01	101)$_2$	= 15)$_8$
Número octal	→	1	5	

EJEMPLO A.12

537,24)$_8$ =	101	011	111	,	010	110	= 101011111,01)$_2$
cifras octales →	5	3	7		2	6	
175,22)$_8$ =	001	111	101	,	010	010	= 1111101,0101)$_2$
cifras octales →	1	5					

Para **transformar un número de octal a decimal** se aplica la expresión [A.3] con $b = 8$. Para **transformar un número entero de decimal a octal** se hacen sucesivas divisiones enteras del número y los subsiguientes cocientes entre ocho (tal como en binario se hacía entre 2). Para transformar la parte fraccionaria de un número decimal a octal se hacen sucesivas multiplicaciones por ocho (como para pasar a binario se hacía por 2).

EJEMPLO A.13

Pasar a decimal el número octal $1.367,25)_8$ *y a octal el número decimal 760,33.*

$1.367,25)_8 = 759,328125)_{10}$; *en efecto:*

$$1.367,25)_8 = 1 \cdot 8^3 + 3 \cdot 8^2 + 6 \cdot 8^1 + 7 \cdot 8^0 + 2 \cdot 8^{-1} + 5 \cdot 8^{-2} =$$
$$= 512 + 192 + 48 + 7 + 0,25 + 0,078128 = 759,328125)_{10}$$

$760,33)_{10} = 1.370,2507)_8$; *en efecto:*

Parte entera:

```
760 |_8_
 45  95 |_8_
  0  15  11 |_8_
      7   3   1
```

b) parte fraccionaria:

0,33	0,64	0,12	0,96 ...
x 8	x 8	x 8	x 8
2,64	**5**,12	**0**,96	7,68 ...

A.4.3 Transformaciones con la base hexadecimal

Los valores binarios y decimales correspondientes a las cifras hexadecimales se muestran en la Tabla A.7.

Tabla A.7. *Cifras hexadecimales y sus correspondientes valores decimal y binario.*

Cifra	Valor decimal	Número binario
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Al ser $b = 16 = 2^4$, podemos hacer las conversiones de binario a hexadecimal y viceversa en forma análoga al sistema octal. Ahora bien, aquí utilizaremos grupos de 4 bits en lugar de grupos de 3 bits.

EJEMPLO A.14

Se verifica:

$$10\ 0101\ 1101\ 1111,\ 1011\ 101)_2 = 25DDF,BA)_H$$

ya que:

Número binario →	0010	0101	1101	1111	,	1011	1010)$_2$ = 25DBF,BA)$_{16}$
Número hexadecimal →	2	5	D	F	,	B	A

también:

$$1ABC70,C4)_H = 1\ 1010\ 1011\ 1100\ 0111\ 0000,1100\ 01)_2$$

En efecto:

Número hexadecimal →	1	A	B	C	7	0	,	C	4
Número binario →	0001	1010	1011	1100	0111	0000	,	1100	0100

De la misma forma que manualmente es muy fácil convertir números de binario a octal, y viceversa, y de binario a hexadecimal y viceversa, también resulta sencillo efectuar esta

operación electrónicamente o por programa, por lo que a veces el computador utiliza este tipo de notaciones intermedias internamente o como entrada/salida.

Para transformar un número de hexadecimal a decimal se aplica la expresión [A.3] con $b = 16$. Para pasar un número de decimal a hexadecimal se hace de forma análoga a los casos binario y octal: la parte entera se divide entre 16, así como los sucesivos cocientes enteros, y la parte fraccionaria se multiplica por 16, así como las partes fraccionarias de los sucesivos productos.

EJEMPLO A.15

Obtener el equivalente decimal del número hexadecimal $A798C{,}1E)_H$. *(Consultar la Tabla A.7 para ver la correspondencia entre símbolos hexadecimales y valores decimales).*

$$\begin{aligned} A798C{,}1E)_H &= (10) \cdot 16^4 + 7 \cdot 16^3 + 9 \cdot 16^2 + 8 \cdot 16 + 12 + 16^{-1} + (14) \cdot 16^{-2} = \\ &= 655.360 + 28.672 + 2.304 + 128 + 12 + 0{,}0625 + 0{,}0546875 = \\ &= 686.476{,}1171)_{10} \end{aligned}$$

Obtener el equivalente hexadecimal del número decimal 4373,79

$$4373{,}79)_{10} \approx 11DD{,}CA3D)_H$$

En efecto,

parte entera:

4573	16		
137	285	16	
093	125	17	16
13	**13**	**1**	**1**

parte fraccionaria:

0,79	0,64	0,24	0,84	....
x 16	x 16	x 16	x 16	
12,64	10,24	**3**,84	**13**,44	....

APÉNDICE **B**

Algunas medidas de uso común en informática

Múltiplos y submúltiplos establecidos para medidas por el SI (*Système Internacional d'Unités*[1]).

MÚLTIPLOS				SUBMÚLTIPLOS		
Prefijo	**Símbolo**	**Factor decimal**	**Factor binario (Nº de Bytes o bits)**	**Prefijo**	**Símbolo**	**Factor decimal**
Kilo-	k-	$1000^1 = 10^3$	$1024^1 = 2^{10} = 1.024$	mili-	m-	$1000^{-1} = 10^{-3}$
Mega-	M-	$1000^2 = 10^6$	$1024^2 = 2^{20} = 1_1048.576$	micro-	µ-	$1000^{-2} = 10^{-6}$
Giga-	G-	$1000^3 = 10^9$	$1024^3 = 2^{30} = 1.073_1741.824$	nano-	n-	$1000^{-3} = 10^{-9}$
Tera-	T-	$1000^4 = 10^{12}$	$1024^4 = 2^{40} = 1_2099.511_1627.776$	pico-	p-	$1000^{-4} = 10^{-12}$
Peta-	P-	$1000^5 = 10^{15}$	$1024^5 = 2^{50} = 1.125_2899.906_1842.624$	femto-	f-	$1000^{-5} = 10^{-15}$
Exa-	E-	$1000^6 = 10^{18}$	$1024^6 = 2^{60} = 1_3152.921_2504.606_1876.976$	atto-	a-	$1000^{-6} = 10^{-18}$
Zetta-	Z-	$1000^7 = 10^{21}$	$1024^7 = 2^{70} = 1.180_3591.620_2717.411_1303.424$	zepto-	z-	$1000^{-7} = 10^{-21}$
Yotta-	Y-	$1000^8 = 10^{24}$	$1024^8 = 2^{80} = 1_4208.925_3819.614_2629.174_1706.176$	yoeto-	y-	$1000^{-8} = 10^{-24}$

Múltiplos de frecuencia (Hz = ciclo/segundo)		
Kilohercio	KHz	10^3
Megahercio	MHz	10^6
Gigahercio	GHz	10^9
Terahercio	THz	10^{12}
Petahercio	PHz	10^{15}

Submúltiplos de tiempo (segundos)		
milisegundo	ms	10^{-3}s
microsegundo	µs	10^{-6}s
nanosegundo	ns	10^{-9}s
picosegundo	ps	10^{-12}s
femtosegundo	fs	10^{-15}s
attosegundo	as	10^{-18}s

1 pie (*feet*) = 12 pulgadas = 30,48 cm
1 pulgada (*inche*) = 1'' = 2,54 cm

[1] Ver: http://www.bipm.fr/fr/si/prefixes.html.

APÉNDICE C

Principales códigos de entrada/salida

Conjunto de caracteres ASCII (ANSI-X3.4, 1968) (7 bits)

		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
00	**0**	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
10	**16**	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
20	**32**	SP	!	“	#	$	%	&	‘	(	)	*	+	,	-	.	/
30	**48**	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	**64**	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	**80**	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
60	**96**	`	a	b	c	d	e	f	g	h	i	j	k	L	m	n	o
70	**112**	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Ejemplo: el carácter “k” es: 60 + B = $6B)_H$ = $110\ 1011)_2$. Su valor decimal es: 96 + 11 = $107)_{10}$.

Caracteres de control ASCII

Hex	Nombre	Descripción
00	NUL	Nulo
01	SOH	Inicio de cabecera
02	STX	Inicio de texto
03	ETX	Final de texto
04	EOT	Fin de transmisión
05	ENQ	Solicitud, petición
06	ACK	Acuse de recibo
07	BEL	Pitido
08	BS	Retroceso de un espacio
09	HT	Tabulación horizontal
0A	LF	Alimentación de línea
0B	VT	Tabulación vertical
0C	FF	Alimentación de hoja
0D	CR	Retorno de carro (línea)
0E	SO	Fuera de código
0F	SI	Dentro de código
10	DLE	Escape de enlace de datos

Hex	Nombre	Descripción
11	DC1	Control de dispositivo 1
12	DC2	Control de dispositivo 2
13	DC3	Control de dispositivo 3
14	DC4	Control de dispositivo 24
15	NAK	Acuse de recibo negativo
16	SYN	Sincronización
17	ETB	Fin de bloque de transmisión
18	CAN	Cancelar
19	EM	Final de soporte (de cinta, etc.)
1A	SUB	Sustituir
1B	ESC	Escape
1C	FS	Separador de archivo
1D	GS	Separador de grupo
1E	RS	Separador de registro
1F	US	Separador de unidad
7F	DEL	Eliminar, borrar, deshabilitar

Conjunto de caracteres ISO 8859-1 (Latín 1)

		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
00	**0**	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
10	**16**	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
20	**32**	SP	!	“	#	$	%	&	‘	(	)	*	+	,	-	.	/
30	**48**	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	**64**	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	**80**	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
60	**96**	`	a	b	c	d	E	f	g	h	i	j	k	L	m	N	o
70	**112**	p	q	r	s	t	U	v	w	x	Y	z	{	|	}	~	DEL
80	**128**																
90	**144**																
A0	**160**		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	–	®	¯
B0	**176**	°	±	²	³	´	µ	¶	•	¸	¹	º	»	¼	½	¾	¿
C0	**192**	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D0	**208**	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E0	**224**	à	á	â	ã	ä	Å	Æ	ç	è	é	ê	ë	Ì	í	Î	ï
F0	**240**	ð	ñ	ò	ó	ô	Õ	Ö	÷	ø	ù	ú	û	Ü	ý	Þ	ÿ

Ejemplo: el carácter “k” es: 60 + B = $6B)_H$ = $0110\ 1011)_2$. Su valor decimal es: 96 + 11 = $107)_{10}$.

Conjunto de caracteres EBCDIC

		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
00	**0**	NUL	SOH	STX	ETX	PF	HT	LC	DEL			SMM	VT	FF	CR	SO	SI
10	**16**	DLE	DC1	DC2	TM	RES	NL	BS	IL	CAN	EM	CC	CU1	IFS	IGS	IRS	IUS
20	**32**	DS	SOS	FS		BYP	LF	ETB	ESC			SM	CU2		ENQ	ACK	BEL
30	**48**			SYN		PN	RS	UC	EOT				CU3	DC4	NAK		SUB
40	**64**			â	Ä	à	á	ã	å	ç	ñ	¢	.	<	(	+	|
50	**80**	&	é	ê	Ë	è	í	î	ï	ì	ß	!	$	*	)	;	¬
60	**96**	-	/	Â	Ä	À	Á	Ã	Å	ç	Ñ	¦	,	%	_	>	?
70	**112**	ø	É	Ê	Ë	È	Í	Î	Ï	Ì	`	:	#	@	‘	=	“
80	**128**	Ø	a	b	C	d	e	f	g	H	i	«	»	õ	ý	Þ	±
90	**144**	°	j	k	L	m	n	o	p	Q	r	ª	º	æ	¸	Æ	¤
A0	**160**	µ	~	s	T	u	v	w	x	y	z	¡¡	¿	Đ	[	þ	®
B0	**176**	¬	£	¥	.	©	§	¶	¼	½	¾	Ý	¨	¯	]	´	×
C0	**192**	{	A	B	C	D	E	F	G	H	I	-	ô	ö	ò	ó	õ
D0	**208**	}	J	K	L	M	N	O	P	Q	R	¹	û	ü	ù	ú	Ý
E0	**224**	\	÷	S	T	U	V	W	X	Y	Z	²	Ô	Ö	Ò	Ó	Õ
F0	**240**	0	1	2	3	4	5	6	7	8	9	³	Û	Ü	Ù	Ú	

Ejemplo: el carácter “k” es: $90 + 2 = 92)_H = 0001\ 0010)_2$. Su valor decimal es: $144 + 2 = 146)_{10}$.

Caracteres de control EDCDIC

Hex	Carácter	Significado
00	NUL	Nulo
01	SOH	Comienzo de cabezera
02	STX	Comienzo de texto (TeXt)
03	ETX	Final de texto TeXT
04	PF	Desconectar perforadora
05	HT	Tabulación horizontal
06	LC	Minúsculas
07	DEL	Eliminar, borrar
0A	SMM	Inicio de mensaje manual
0B	VT	Tabulación vertical
0C	FF	Alimentación de hoja
0D	CR	Retorno de carro
0E	SO	Fuera de código
0F	SI	Dentro de código
10	DLE	Escape de enlace de datos
11	DC1	Dispositivo de control 1
12	DC2	Dispositivo Control 2
13	TM	Marca de cinta
14	RES	Restaurar, restablecer
15	NL	Pasar a línea siguiente
16	BS	Retroceso de un espacio
17	IL	Sin función
18	CAN	Cancelar
19	EM	Final del soporte de información
1A	CC	Control del cursor
1B	CU1	Usuario cliente 1

Hex	Carácter	Significado
1C	IFS	Separador para intercambio de archivos
1D	IGS	Separador para intercambio de grupos
1E	IRS	Separador para intercambio de registros
1F	IUS	Separador para intercambio de unidades
20	DS	Selección de dígito
21	SOS	Comienzo de significado
22	FS	Separador de campos
24	BYP	Desviar
25	LF	Alimentación de línea
26	ETB	Final de transmisión de bloque
27	ESC	Escape
2A	SM	Establecer modo
2B	CU2	Usuario cliente 1
2D	ENQ	Solicitar
2E	ACK	Reconocimiento
2F	BEL	Emisión de sonido (pitido)
32	SYN	Sincronización inactiva
34	PN	Conectar perforadora
35	RS	Parar lectora
36	UC	Letras mayúsculas
37	EOT	Final de transmisión
3B	CU3	Usuario cliente 3
3C	DC4	Dispositivo de control 4
3D	NAK	reconocimiento negativo
3F	SUB	Sustituir

Conjunto de caracteres UNICODE

Utiliza n = 16 bits para representar 65.536 símbolos posibles, comprendidos en un rango que va de 0000 a FFFF.

Zona		Códigos	Símbolos codificados	Nº de caracteres	
A	0000	0000 a 007F	Latín Básico (00 a 7F), definido en la norma ASCII ANSI-X3.4	128	8.192
		0080 a 00FF	Suplemento Latín-1 (ISO 8859-1)	128	
		0100 a 017F	Ampliación A de Latín	128	
		0180 a 024F	Ampliación B del Latín	80	
		0250 a 02AF	Ampliación del Alfabeto Fonético Internacional (IPA)	96	
		02BF a 02FF	Espaciado de letras modificadoras	65	
		0300 a 036F	Combinación de marcas diacríticas (tilde, acento grave, etc.)	112	
		0370 a 03FF	Griego	144	
		0400 a 04FF	Cirílico	256	
		0530 a 058F	Armenio	96	
		0590 a 05FF	Hebreo	112	
		0600 a 06FF	Árabe	256	
		0700 a 074F	Sirio	80	
			etc.		
	3FFF	2000 a 3FFF	Símbolos generales y caracteres fonéticos chinos, japoneses y coreanos		8.192
I	4000 9FFF		Ideogramas		24.576
O	A000 DFFF		Pendiente de asignación		16.384
R	E000 FFFF		Caracteres locales y propios de los usuarios Compatibilidad con otros códigos		8.192

APÉNDICE D

Principales siglas y acrónimos utilizados en el presente libro

A/D
Analógico/Digital

ACM *(Association of Computing Machinery)*

AC-PDP *(AC plasma display panel)*
Panel de visualización de plasma AC

ADF *(Advanced Disk File)*
(Unidad de disco de IBM)

ADSL *(Asymmetric digital subscriber line)*
Línea de abonado digital asimétrica

AGP *(Advanced Graphics Port)*
Puerto avanzado para gráficos

ALGOL *(ALGOrithmic Language)*
Lenguaje algorítmico

ALU *(Arithmetic Logic Unit)*
Unidad aritmético-lógica

AM *(Amplitude Modulation)*
Modulación de amplitud

AM-LCD *(Active-Matrix LCD)*
Pantalla de cristal líquido activa

ANSI *(American National Standars Institute)*
Instituto nacional americano de normalizaciones

AR *(Address Register)*
Registro de dirección

ARPANET *(Advanced Research Projects Agency Network)*

ASCII *(American Standard Code for Information Interchange)*
Código estándar americano para intercambio de información

ASIC *(Application Specific Integrated Circuit)*
Circuito integrado de aplicación específica

ASK *(Amplitude shift keying)*
Modulación por desplazamiento de amplitud

ASM *(Algorithm-State-Machine)*
Máquina de estados algorítmica

AT *(Advanced Technology)*
Tecnología avanzada (versión de PC: PC-AT)

ATA *(Advanced Technology Attachment)*
Adjunto a PC-AT

ATM *(Asynchronous Transfer Mode)*
Modo de transferencia asíncrono

BCD *(Binary Coded Decimal)*
Dígitos decimales codificados en binario

BEDO RAM (*Burst extended Data Output RAM)*
RAM con salida de datos a ráfagas

BIOS *(Basic Input Output System)*
Sistema Básico de Entrada/Salida

BISDN *(Broadband ISDN)*
RDSI de banda ancha (RDSIBA)

BISYNC *(Binary Synchronous Communications)*
Comunicaciones síncronas binarias

Bit *(BInary y digiT)*
Bit (dígito binario)

BMP *(BitMap)*
Mapa de bits

BNF *(Backus-Naur Form)*
Notación Backus-Naur

BOT *(Boot Of Tape)*
Comienzo de cinta

BRZ *(Bipolar Return to Zero)*
Bipolar con retorno a cero (codificación)

CAD *(Computer Aided Design)*
Diseño con ayuda de computadora

CADMAT *(Computer Aided Design Manufacturing and Testing)*
Diseño, fabricación y test con ayuda de computadora

CAI *(Computer Assisted Instruction)*
Enseñanza con ayuda de computadora
CAL *(Computer Aided Learning)*
Enseñanza con ayuda de computadora
CAM *(Computer Aided Manufacturing)*
Fabricación con ayuda de computadora
CAMAC *(Computer Automated Measurement and Control)*
Medidas y control automatizados para computadoras (bus estándar)
CAV *(Constant Angular Velocity)*
Velocidad angular constante
CCD *(Charge-Coupled Device)*
Dispositivo acoplado por carga
CCITT *(Consultative Committee for International Telegraphy and Telephony)*
Comité consultivo para telegrafía y telefonía internacionales
CD *(Compact Disk)*
Disco compacto
CDMA
Controlador para DMA
CD-R *(CD Recordable)*
CD grabable
CDRAM *(Cache DRAM)*
RAM dinámica con caché
CD-ROM *(Compact Disc, Read Only Memory)*
Disco compacto de sólo lectura
CD-RW *(CD-ReWritable)*
CD regrabable
CE *(Consumer Electronics)*
Electrónica de consumo (Windows CE)
CGA *(Color Graphics Adapter)*
Adaptador para gráficos en color
CGI *(Common Gateway Inteface)*
Interfaz común de pasarela
CISC *(Complex Instruction Set Computer)*
Computadora con repertorio de instrucciones complejo
CLV *(Constant Lineal Velocity)*
Velocidad lineal constante
COBOL *(COmmon Bussines Oriented Language)*
Lenguaje orientado a gestión usual
CODE
COmputador Didáctico Elemental
codop
Código de operación.
COW *(Cluster of Workstation)*
Cluster de estaciones de trabajo
cps
caracteres por segundo
CPU *(Central Processing Unit)*
Unidad de procesamiento central. Procesador
CRC *(Cyclic redundancy check)*
Comprobación de redundancia cíclica
CRTC *(Catode Ray Tube Controler)*
Controlador del tubo de rayos catódicos
CSMA *(Carrier sensing multiple access)*
Acceso múltiple sensible a portadora
CSMA-CD *(Carrier sensing multiple access with collision detection)*
Acceso múltiple sensible a portadora con detección de colisión
CTR *(Catode Ray Tube)*
Tubo de rayos catódicos
CU *(Control Unit)*
Unidad de control
CYMK *(Cyan, Yelow, Magenta, blacK)*
Azul, amarillo, rojo y negro

D/A
Digital/Analógico
DAT *(Digital Audio Tape)*
Cinta de audio digital
DBMS *(Database Management System)*
Sistema de gestión de la base de datos
DCE *(Data circuit terminating equipment)*
Equipo terminal del circuito de datos
DDL *(Data Description Language)*
Lenguaje de descripción de datos
DDR DRAM *(Double-Data Rate)*
DRAM de doble velocidad de datos
DDS *(Digital Data Storage)*
Almacenamiento digital de datos (estándar grabación CD)
DEC *(Digital Equipment Corporation)*
DFD
Diagrama de flujo de datos
DIMM *(Dual In-line Memory Module)*
Módulo de memoria con conexiones alineadas en los dos laterales
DMA *(Direct Memory Access)*
Acceso directo a memoria
DML *(Data Manipulation Language)*
Lenguaje de manipulación de datos (en bases de datos)
DNS *(Domain Name System)*
Sistema de nombres de dominio
DoD *(Departament of Defense)*
Departamento de Defensa (USA)
dpi *(dot per inche)*
puntos por pulgada

DR *(Data Register)*
Registro de datos
DRAM *(Dynamic RAM)*
RAM dinámica
DTE *(Data terminal equipment)*
Equipo terminal de datos
DVD *(Digital Versatil Disk)*
Disco digital versátil
DVD-R *(DVD Recordable)*
DVD grabable
DVD-RAM
DVD con acceso aleatorio
DVD-RW *(CD-ReWritable)*
DVD regrabable
DXF *(Document eXchange Format)*
Formato para intercambio de documentos

E
Entrada
E/S
Entrada/Salida
EAN *(European Article Numbering)*
Numeración europea de artículos
EBCDIC *(Extended Binary Coded Decimal Interchange Code)*
Código de intercambio BCD ampliado
e-comerce *(Electronic comerce)*
Comercio electrónico
EDO RAM (*Extended Data Out Dynamic RAM)*
RAM con salida de datos ampliada
EDRAM *(Enhanced DRAM)*
RAM dinámica mejorada
EDSAC *(Electronic Storage Automatic Calculator)*
EEPROM *(Electrical Erasable PROM)*
PROM borrable eléctricamente
EGA *(Enhanced Graphics Adapter)*
Adaptador para gráficos mejorado
ei
Elemento de imagen (píxel)
EIDE *(Enhaced Integrated Drive Electronics)*
Electrónica de controlador integrada y mejorada
EISA *(Extended Industry Standard Architecture)*
Arquitectura estándar ampliada para la industria
e-mail *(Electronic mail)*
Correo electrónico
ENIAC *(Electronic Numerical Integrator and Computer)*
EOT *(End Of Tape)*
Fin de cinta
EPROM *(Erasable PROM)*
PROM borrable
EPS *(Encapsulated Poscript)*
Poscript encapsulado

FCFS *(First-Come First-Served)*
Primero en llegar, primero en ser atendido
FDDI *(Fiber Distributed Data Interface)*
Interfaz de distribución de datos con fibra
FDM *(Frequency-division multiplexing)*
Multiplexación por división en frecuencias (FDM)
FEC *(Forward Error Correction)*
Corrección de errores anticipada
FED *(Field-emissions display)*
Pantalla de emisión de campo
FF *(Flag Flip-flop)*
Biestables indicadores
FIFO *(First In, First Out)*
Primero en entrar, primero en salir
F-LCD *(Ferroelectric LCD)*
Pantalla de cristal líquido ferroeléctrica
FM *(Frequency Modulation)*
Modulación de frecuencia
FORTRAN *(FORmula TRANslator)*
Traductor de fórmulas (lenguaje de programación)
FPD *(Flat-Planel Display)*
Pantalla plana
FPGA *(Field Programmable Gate Array)*
Matriz de puertas programables por campo
FPM RAM (*Fast Page Mode)*
RAM con modo de página rápida
FPU *(Float Point Unit)*
Procesador de coma flotante
FSK *(Frequency shift keying)*
Modulación por desplazamiento en frecuencia
FSM *(Finite-State-Machine)*
Máquina de estados finitos
FSMD *(FSM with Datapaph)*
Máquina de estados finitos con camino de datos
FTP *(File Transfer Protocol)*
Protocolo de transferencia de archivos (FTP)
GDT *(Global Descriptor Table)*
Tabla de descriptores globales (MV en el Pentium)
GIF *(Graphic Interchange Format)*
Formato para intercambio de gráficos
GPIB *(General Purpose Interface Bus)*
Bus de interfaz de uso general

GUI *(Graphical User Interface)*
Interfaz gráfica de usuario
HDL *(Hardware Description Language)*
Lenguaje de descripción del hardware
HDLC *(High level Data Link Control)*
Control de enlace de datos de alto nivel
HFS *(Hierarchical File System)*
Sistema jerárquico de archivos (estándar de grabación de CD)
HGC *(Hercules Graphic Card)*
Tarjeta de gráficos Hércules
HRRN *(Highest Response Ratio Next)*
Turno para el de mayor coeficiente de respuesta
HTML *(Hypertext Markup Language)*
Lenguaje de marcas para hipertextos
HTTP *(Hypertext Transfer Protocol)*
Protocolo de transferencia de hipertextos

IA *(Interruption Acknowledge)*
Reconocimiento de la interrupción
IBM *(International Bussines Machines)*
ICMP *(Internet Control Message Protocol)*
Protocolo de mensajes de control en Internet
IDE *(Integrated Drive Electronics)*
Electrónica de controlador integrada
IEEE *(Institute of Electrical and Electronics Engineering)*
IGES *(Initial Ghaphics Exchange Specification)*
Especificación inicial para intercambio de gráficos
IKBS *(Intelligent Knowledge-Based Systems)*
Sistemas expertos o basados en el conocimiento
InterNIC *(Internet Network Information Center)*
Centro de información de red Internet
IOCS *(Input-Output Control System)*
Sistema de control de E/S
IOP *(Input/Output Processor)*
Procesador de E/S
IP *(input port)*
Puerto de entrada
IP *(Internet Protocol)*
Protocolo internet
IR *(Interruption Request)*
Petición de interrupción
IR *(Intruction Register)*
Registro de instrucción
IRC *(Internet Relay Chat)*
Tertulias Internet o chats
IRG *(Inter Record Gap)*
Interbloque entre registros (en cintas magnéticas)
ISDN *(Integrated Services Digital Network)*
Red digital de servicios integrados (RDSI)
ISO *(International Organization for Standardization)*
Organización Internacional de normalizaciones
ISR *(Interruction Service Routine)*
Rutina de servicio de interrupción

JPEG *(Joint Photograph Expert Group)*
Asociación de grupos de expertos en fotografía
JVM *(Java Virtual Machine)*
Máquina virtual Java

LAN *(Local Area Network)*
Red de área local
LAN
Lenguaje de alto nivel
LCD *(Liquid Crystal Display)*
Pantalla de cristal líquido
LCR *(Longitudinal Redundancy Checking)*
Comprobación de redundancia longitudinal
LDT *(Local Descriptor Table)*
Tabla de descriptores locales (MV en el Pentium)
LE
Lenguaje ensamblador
LM
Lenguaje máquina
lpm
líneas por minuto
LRU *(Least Recently Used)*
Menos recientemente usado
LSB *(Least Significative Bit)*
Bit menos significativo
LSI *(Largue Scale Integration)*
Integración en gran escala
LUT *(Look-Up Table)*
Tabla de consulta

M
Memoria
M^2FM *(Modified-Modified FM)*
Modulación de frecuencia modificada-modificada
maxterm
Término máximo
MCA *(Micro-Channel Architecture)*
Arquitectura microcanal
MDA *(Monochrome Display Adapter)*
Adaptador monocromo de pantalla

MFLOPS
Millones de instrucciones de coma flotante por segundo
MFM *(Modifed FM)*
FM modificada
MFT *(Master File Table)*
Tabla maestra de archivos (en Windows NT)
MIDI *(Musical Instrument Digital Interface)*
Interfaz digital para instrumentos musicales
MIMD *(Multiple Instruction stream Multiple Data stream)*
Computadoras de múltiples flujos de instrucciones y de datos
minterm
Término mínimo
MIPS
Millones de instrucciones por segundo
MISD *(Multiple Instruction stream Single Data stream)*
Computadoras de múltiples flujos de instrucciones y único flujo de datos
MMU *(Memory Management Unit)*
Unidad de gestión de memoria
MNRZI *(Modified NRZI)*
NRZI modificado (codificación)
MO *(Magnetic-Optic)*
Magneto-óptico (disco)
MP
Memoria principal
MPEG *(Motion Picture Expert Group)*
Grupo de Expertos en Imágenes en Movimiento (MPEG)
MPP *(Massively Parallel Processors)*
Procesadores masivamente paralelos
MSB *(Most Significative Bit)*
Bit más significativo
MS-DOS *(Microsoft Disk Operating System)*
Sistema operativo de discos de Microsoft
MSI *(Medium Scale Integration)*
Integración en media escala
MV
Memoria Virtual
MVS *(Múltiple Virtual Storage)*
Memoria virtual múltiple (IBM)
MZR *(Magnetic Zoned-Recording)*
Grabación magnética por zonas

NaN *(No a Number)*
No es un número
NFS *(Network File System)*
Sistema de archivos en red
NIC *(Network interface card)*
Tarjeta de interfaz con la red
NOW *(Network of Workstation)*
Redes de estaciones de trabajo
NRU *(Not Recently Used)*
No usado recientemente
NRZ *(Nonreturn-to- Zero)*
Sin retorno a cero (codificación)
NRZI *(Non Return to Zero Invert)*
Sin retorno a cero inverso (codificación)
N-S *(Nassi-Schneiderman)*
Nassi-Schneiderman (diagramas de N-S)
NT *(New Technology)*
Nueva Tecnología (Windows NT)
NTFS *(NT File System)*
Sistema de archivos de nueva tecnología

OC-n *(Optical carrier level-n)*
Portadora óptica de nivel-n
OCR *(Optical Character Recognition)*
Reconocimiento óptico de caracteres
ODE
Ordenador Didáctico Elemental
OMR *(Optical Mark Reader)*
Lector óptico de marcas
OOK *(On-Off Keyed)*
Modulación binaria
OP *(Output Port)*
Puerto de salida
OSI *(Open Systems Interconection)*
Interconexión de sistemas abiertos

PAL *(Programmable Array Logic)*
Lógica con matriz programable
PA-LC *(Plasma-addressed liquid crystal)*
Cristal líquido direccionable por plasma
PC *(Personal Computer)*
Computadora personal
PC *(Programm Counter)*
Contador del programa
PCI *(Peripheral Component Interconected)*
Interconexión de componentes periféricos (estándar de bus)
PCM *(Pulse Code Modulation)*
Modulación por codificación de pulsos
PDA *(Personal digital assistant)*
Asistente digital personal
PDM *(Pulse Duration Modulation)*
Modulación por anchura de pulso
PDU *(Protocol Data Unit)*
Unidad de datos del protocolo

PE *(Phase Encoding)*
Codificación en fase
PIC *(Programmable Interrupt Controller)*
Controlador de interrupciones programable
PICT *(PICTure)*
Pintura, cuadro
PLA *(Programmable Logic Array)*
Array lógico programable
PM-LCD *(Passive-Matrix LCD)*
Pantalla de cristal liquido de matriz pasiva
PNG *(Portable Network Graphics)*
Gráficos de red compatibles (portátiles)
POS *(Point Of Sale)*
Punto (o puesto) de ventas
POST *(Power On Self Test)*
Autodiagnóstico de encendido
PPM *(Pulse Position Modulation)*
Modulación por posición de pulso
ppm
páginas por minuto
PR *(Pulse Ratio)*
Coeficiente de pulso (codificación)
PROM *(Programmable ROM)*
Memoria de sólo lectura programable
PSK *(Phase shift keying)*
Modulación por desplazamiento de fase
PWM *(Pulse Wide Modulation)*
Modulación por anchura de pulso

QAM *(Quadrature Amplitude Modulation)*
Modulación de amplitud en cuadratura

RAID *(Redundant Array of Independent Disks)*
Agrupación redundante de discos independientes
RAM *(Ramdom Acccess Memory)*
Memoria de acceso aleatorio
RAMAC *(Ramdom Access Method of Accounting and Control)*
(Primer disco, de IBM)
RDSI
Red Digital de Servicios Integrados
RDSIBA
RDSI de banda ancha
RF *(Register File)*
Banco de registros
RGB *(Read, Green, Blue)*
Rojo, Verde, Azul
RIMM *(Rambus In line Memory Module)*
Modulo de memoria Rambus con conexiones alineadas
RISC *(Reduced Instruction Set Computer)*
Computadora con repertorio de instrucciones reducido
ROM *(Read Only Memory)*
Memoria de sólo lectura
rpm
Revoluciones (vueltas) por minuto
RS *(Recommended standard)*
Estándar recomendado
RS *(Return to Saturation)*
Vuelta a saturación (codificación con)
RT
Registro temporal
RTL *(Register Transfer Level)*
Nivel de transferencia entre registros
RTP *(Real Time Transport Protocol)*
Protocolo de transporte en tiempo real
RZ *(Return to Zero)*
Vuelta a cero (codificación con)

S
Salida
SAGE *(Semi-Automatic Ground Enviroment)*
SCSI *(Small Computer System Interface)*
Interfaz para pequeñas computadoras
SDRAM *(Synchronous Dynamic RAM)*
RAM dinámica síncrona
SIMD *(Single Instruction stream Multiple Data stream)*
Computadoras de un único flujo de instrucciones y múltiples flujos de datos
SIMM *(Single In-line Memory Module)*
Módulo de memoria con conexiones laterales en línea
SISD *(Single Instruction stream Single Data stream)*
Computadoras de un único flujo de instrucciones y un único flujo de datos
SMP *(Symetric Multiprocessing)*
Multiprocesamiento simétrico
SMTP *(Simple Mail Transfer Protocol)*
Protocolo sencillo de transferencia de correo electrónico
SNR *(Signal to noise ratio)*
Relación señal-ruido
SO
Sistema Operativo
SP *(Stack Pointer)*
Puntero de pila
SPEC *(System Performance Evaluation Cooperative)*
Conjunto para evaluación del rendimiento de computadoras
SPN *(Shortest Process Next)*
Turno para el de menor tiempo de procesador

SPOOL *(Simultaneous Peripherical Operation Off Line)*
Funcionamiento simultáneo de periféricos fuera de línea
SQL *(Structured Query Language)*
Lenguaje estructurado de consulta de datos
SRAM *(Static RAM)*
RAM estática
SRT *(Sortest Remaining Time)*
Turno para el que le queda menos tiempo de procesador
SSD *(Solid State Disks)*
Discos de estado sólido
SSF *(Sortest Seek First)*
El más cercano se atiende primero
SSI *(Small Scale Integration)*
Integración en pequeña escala
STN-LCD *(Supertwist nematic LCD)*
Pantalla de cristal líquido de efecto de campo STN
SVGA *(Super Video Graphics Array)*
VGA superior
SXGA *(Super eXtended Graphics Array)*
Matriz gráfica ampliada y superior

TCAL *(Thermal recalibration)*
Recalibración térmica
TDM *(Time division multiplexing)*
Multiplexación por división en el tiempo
TFT *(Thin-Film Transistor)*
Transistor de película delgada
TFT-LCD *(Thin Film Transistor LCD)*
Pantalla de cristal líquido con transistores de película delgada
TIFF *(Tagged Image File Formats)*
Formato de archivos de imágenes etiquetadas
TN-LCD *(Twisted nematic field effect LCD)*
Pantalla de cristal líquido de efecto de campo TN
TV
Televisión

UDP *(User Datagram Protocol)*
Protocolo de datagrama de usuario
UMA *(Uniform Memory Access)*
Acceso uniforme a memoria (multiprocesadores)
UNIVAC *(UNIVersal Automatic Computer)*
URL *(Uniform resource locator)*
Localizador uniforme de recursos
USB *(Universal Serial Bus)*
Bus serie universal

VCR *(Vertical Redundancy Checking)*
Comprobación de redundancia vertical
VGA *(Video Graphics Array)*
Matriz gráfica para video
VHDL *(Very High speed integrated circuits HDL)*
HDL para circuitos integrados de muy alta velocidad
VLSI *(Very Largue Scale Integration)*
Integración a muy gran escala
VRAM *(Video RAM)*
Memoria de vídeo

WAIS *(Wide Area Information Service)*
Servicio de información de área mundial.
WAN *(Wide Area Network)*
Red de área amplia
WAP *(Wireless Application Protocol)*
Protocolo para comunicaciones inalámbricas
WDM *(Wavelength division multiplexing)*
Multiplexación por división en longitud de onda
WWW *(World Wide Web)*
Red mundial (world wide web o www)

XGA *(Extended Graphics Array)*
Adaptador para gráficos ampliado
XVGA *(Extended Video Graphics Array)*
Matriz gráfica para vídeo ampliada

ZBR *(Zoned-bit recording)*
Grabación por pequeñas zonas
ZCAV *(Zoned-Constant Angular Velocity)*
Velocidad angular constante por zonas

Índice analítico

D

E

N

O

T

U

V

W

X

Z

INTRODUCCIÓN A LA INFORMÁTICA

ALBERTO PRIETO ESPINOSA
ANTONIO LLORIS RUIZ
JUAN CARLOS TORRES CANTERO

El presente libro viene avalado por ser la cuarta edición de uno de los textos de mayor prestigio, por su rigor y claridad, realizado por autores en lengua española sobre aspectos fundamentales de la Informática. La obra trata de dar al lector unos conocimientos básicos, completamente actualizados, sobre los computadores, tratando tanto el hardware como el software. Para entender profundamente a la Informática es necesario mostrar una visión general de ella, que permita ver cómo sus diferentes partes encajan en un todo.

La cuarta edición mejora la estructura de capítulos de la edición anterior y además incluye los cambios y novedades más significativos que se han producido en los cuatro años que han transcurrido desde ésta. El material incluido en cada uno de los capítulos se ha revisado minuciosamente, incluyendo nuevos conceptos y eliminando los que han dejado de tener interés, actualizando tablas de características, mejorando el contenido y calidad de las figuras, y procurando dar una visión más práctica y aplicada de los distintos temas. El libro se ha estructurado en capítulos más cortos, haciéndolo así más pedagógico, y pudiéndose adaptar mejor a distintos tipos de cursos, en diferentes titulaciones. También se ha ampliado el número de figuras (367), tablas (149), ejemplos (212) y ejercicios (313). Los principales temas que se incluyen son los siguientes:

- Definiciones y conceptos básicos. Aplicaciones de la Informática
- Estructuras de datos
- Algoritmos
- Representación de textos, sonidos e imágenes
- Sistemas digitales combinacionales y secuenciales
- Esquema de funcionamiento de un computador
- Lenguajes máquina y ensamblador
- El computador a nivel de micromáquina (diseño de un computador básico)
- El sistema de memoria
- Periféricos de entrada/salida
- Estructura y arquitectura de computadores
- Conceptos sobre sistemas operativos
- Lenguajes de programación y traductores
- Archivos de datos
- Bases de datos
- El concepto de ingeniería del software
- Transmisión y comunicación de datos
- Redes de computadores e Internet
- Desarrollo y evolución histórica de la Informática

En todo momento se han tratado de enfocar los conceptos presentados de la forma más descriptiva y práctica posible para que el lector pueda comprender satisfactoriamente tanto los sistemas actuales que están en la vanguardia de la tecnología como los futuros.

La obra es útil a nivel universitario en asignaturas de Introducción a la Informática, Informática General, Introducción a los Computadores o Fundamentos de Computadores. También lo es para los profesionales y los usuarios de la Informática que deseen tener una visión actualizada de los temas más básicos de esta disciplina, así como para cualquier persona que tenga interés en adquirir una formación rigurosa en los aspectos más destacados de la Informática.

Para la presente edición se ha diseñado una página web, que incluye abundante material suplementario para uso de estudiantes y profesores. La dirección de acceso es: **http://www.mhe.es/universidad/informatica/introduccion.**

Los contenidos de este libro se complementan con el libro de problemas y ejercicios *Conceptos de Informática, Serie Schaum*, de los autores Alberto y Beatriz Prieto, editado por McGraw-Hill, y con ISBN 84-481-9857-3

www.mcgraw-hill.es
The McGraw·Hill Companies